НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

«На правах рукопису»
УДК 004.04

До захисту допущено:

Завідувач кафедри

_____ Ігор ПАРХОМЕЙ

«___»_____2020 р.

Дипломний проект

на здобуття ступеня магістра

за освітньо-професійною програмою «Інформаційне забезпечення робототехнічних систем»

зі спеціальності 126 «Інформаційні системи та технології»

на тему: «Підсистема обробки даних на мобільній платформі пристроїв господаря розумного будинку»

Виконав:
студент ІІ курсу, групи ІК-91-мп
Серга Олексій Максимович _____

Керівник:
К.т.н., доцент
Мелкумян Катерина Юріївна _____

Консультант з норм. контролю:
к.т.н., доцент
Пасько Віктор Петрович _____

Рецензент:
Директор КБИС, к.т.н., доцент,
Фіногенов Олексій Дмитрович _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

Рівень вищої освіти – другий (магістерський)

Напрям підготовки – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційне забезпечення робототехнічних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Ігор ПАРХОМЕЙ

«___»_____2020 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Сєрзі Олексію Максимовичу

1. Тема дисертації «Підсистема обробки даних на мобільній платформі пристроїв господаря розумного будинку», науковий керівник дисертації Мелкумян Катерина Юріївна, к.т.н., доцент, затверджені наказом по університету від « 26 » жовтня 2020р. № 3132-с

2. Термін подання студентом проекту: 21.12.2020 р.

3. Об’єкт дослідження – є семантичний аналіз ХМЛ-файлу на мобільній платформі Android.

4. Вихідні дані до проекту: бібліотека з реалізованими методами семантичного аналізу, додаток с апробацією цієй бібліотеки, аналіз реалізованих методів, алгоритм з пошуку найоптимальнішого серед запропонованих методів для певного девайсу при певних умовах.

5. Перелік завдань, які потрібно розробити:
   - Проаналізувати існуючі бібліотеки та інструменти парсингу.
   - Запропонувати елемент, за допомогою якого можна обійти усі проблеми сучасних існтрументів.
   - Розробити за допомогою цього елементу нові методи семантичного аналізу.
   - Порівняти реалізовані методи між собою за певними характеристиками.

- Розробити алгоритм для пошуку найоптимальнішого методу.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: Процес холодного старту Android додатку, Алгоритм функціонування додатка, Алгоритм зчитування XML-файла додатком, UML діаграма класів додатка, Життєвий цикл Activity, Алгоритм пошуку найоптимальнішого методу парсингу.

7. Орієнтовний перелік публікацій – «Огляд багатопоточності та бібліотек для роботи з нею на мобільній платформі Android».

8. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Перевірка на співпадіння	доцент Лісовиченко О. І.		
Норм. контроль	доцент Пасько В.П.		

9. Дата видачі завдання: 01.10.2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1	Огляд та аналіз сучасних парсерів XML-файлів для платформи Android.	01.10.19-15.10.19 рр.	
2	Пошук інструментів та бібліотек для створення поліпшених методів обробки XML-файлів.	16.10.19-16.12.19 рр.	
3	Реалізація бібліотеки з поліпшеними методами та інтегрування її у додаток.	17.12.19-17.06.20 рр.	
4	Аналіз реалізованих методів та їх порівняння	18.06.20-30.09.20 рр.	
5	Створення алгоритму пошуку найоптимальнішого методу	01.10.20-10.11.20 рр.	
6	Оформлення дипломного проекту	11.11.20-22.11.20 рр.	
7	Попередній захист	23.11.2020	
8	Нормоконтроль	30.11.20-06.12.20 рр.	
9	Перевірка на співпадіння	01.12.20-10.12.20 рр.	
10	Захист	21.12.2020	

Студент _____ Олексій СЕРГА
(підпис)

Керівник проекту _____ Катерина МЕЛКУМЯН
(підпис)

АНОТАЦІЯ

Дипломна робота виконана на 92 сторінках і містить 36 ілюстрацій, 38 таблиці, 4 формули, 6 додатків. При розробці використано інформацію з 15 джерел.

Метою даної роботи є поліпшення методів семантичного аналізу XML-файла за рахунок створення бібліотеки та апробація її у додатку для плафторми Android.

Метод дослідження аналізує реалізовані методи за наступними параметрами: час повної обробки та ресурсоємкість.

Було розроблено бібліотеку з використанням інструментів, які дозволяють працювати з багатопоточністю, а саме: Threads, RxJava, Kotlin Coroutines. За допомгою першої бібліотеки було реалзовано два методи обробки: в один потік та у два потоки, за допомогою другої – три методи: в один потік, у два потоки та з використання методу *parallelStream()*, за допомогою третьої – три методи: в один потік, в два потоки, в чотири потоки. Після реалізації програмної частини та систематичних тестів було розроблено алгоритм з пошуку найоптимальнішого методу на певному девайсі при певних умовах.

Програмний продукт розроблено у середовищі Android Studio, частково протестовано реалізовані методи також у Android Studio.

Ключові слова: семантична обробка, XML-файл, платформа Android, XMLPullParser, SAXParser, DOMParser, Threads, RxJava, Kotlin Coroutines, Java, Kotlin, багатопоточність, асинхроність, паралельне програмування.

ANNOTATION

The research paper performed at 92 pages and contains 36 illustrations, 38 tables, 4 formulas, 6 appendices. In developing uses information from 15 sources.

The purpose of this work is to improve the methods of semantic analysis of an XML-file by creating a library and testing it in the application for the Android Platform.

The research method analyzes the implemented methods according to the following parameters: time of complete processing and resources intensity.

A library was developed using tools that allow multithreading, such as Threads, RxJava, Kotlin Coroutines. By the support of the first library, two methods of processing were realized: in one thread and in two threads, with the second one there were three methods: in one stream, in two streams and using the *parallelStream()* method, with the third one - three methods: in one thread , in two threads, in four threads. After the implementation of the software and systematic tests, an algorithm was developed to find the best method on a particular device under certain conditions.

The software product is developed in the Android Studio environment, partly tested also in Android Studio.

Keywords: semantic processing, XML-file, Android platform, XMLPullParser, SAXParser, DOMParser, Threads, RxJava, Kotlin Coroutines, Java, Kotlin, multithreading, asynchronous, parallel programming.

Пояснювальна записка
до магістерської дисертації
на тему: «Підсистема обробки даних на мобільній платформі пристроїв господаря розумного будинку»

Київ – 2020 року

ЗМІСТ

СПИСОК СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

XML – Extensible Markup Language (з англ. розширювана мова розмітки)

ОС – Операційна система

МОС – Мобільна операційна система

SAX – Simple API для XML

API – Application programming interface (з англ. програмний інтерфейс додатка)

ВАП – Віртуальний адресний простір

CDATA – Character data (з англ. символьні дані)

DOM – Document Object Model (з англ. об’єктна модель документа)

SQL – Structured query language (з англ. мова структурованих запитів)

ВСТУП

В даний час мобільні операційні системи набули значної популярності. Неспромога уявити собі людину без гаджета, за допомогою якого можна побачити найрізноманітніший контент в інтернеті, подивитися фотографії друзів, поділитися своїми відчуттями і відео в соціальних мережах. А мобільність і зручність використання таких гаджетів – це саме те, що потрібно людям. Смартфон весь час знаходиться в руках людини або в кишені.

Розробка мобільних додатків отримала дуже популярне місце серед програмістів. Гігантські компанії замовляють такий софт постійно. Але оскільки монополісти на ринку мобільних операційних систем повністю керують ними і відповідають за їх оновлення, вони встановлюють правила для мобільного сфота. Одним з таких абстрактних (або необов'язкових) правил є зручність використання і швидкість візуального інтерфейсу.

Тим не менш, є багато завдань, які в даний час неможливо виконати швидко і без величезного навантаження на систему гаджета. Однією з таких проблем є семантичний аналіз (парсинг) великого обсягу даних. Для вирішення цієї проблеми потрібні великі компанії та стартапи, які все більше і більше працюють з такими типами даних.

1. ОГЛЯД СУЧАСНИХ МЕТОДІВ СЕМАНТИЧНОГО АНАЛІЗУ XML-ФАЙЛА

1.1. Загальний огляд об’єкта дослідження

Найпопулярнішою мобільною операційною системою зараз є платформа Android, а гарно структурованим документом, який може містити при собі багато інформації, що підтримується майже усіма застосунками та системами – XML-файл.

1.1.1. Мобільна платформа Android

В даний час мобільні операційні системи придбали дуже велику популярність. Всі їх використовують. Вони розміщені виробниками більшості своїх гаджетів: смартфонів, планшетів, смарт-годинників, фітнес-трекерів, портативних ігрових консолей, ноутбуків, розумних окулярів, окулярів віртуальної реальності, телевізорів. Навіть для своїх розважальних систем в автомобілях виробники віддають перевагу мобільним платформам.

З самого початку створення мобільних операційних систем до сьогоднішнього дня світ бачив багато можливостей. Більшість з них були дуже популярні серед виробників і споживачів, але не всім вдалося вижити в боротьбі за ринок. Таким чином, на сьогоднішній день можна виділити тільки дві мобільні платформи: Андроїд і IOC.

Мобільна операційна система Андроїд створена на ядрі Лінукса і власної імплементації віртуальної машини Java (JVM) від Google. Вона має відкритий код, тому кожен на сьогоднішній день може завантажити його, змінити і використовувати свою версію для своєї мети. Тим не менш, компанії розуміють, що Гугл робить свою систему краще кожен день, тому вони намагаються тільки візуально змінити операційну систему, щоб вона відрізнялася від конкурентів.

Зараз просто неможливо уявити когось без смартфона. З його допомогою ви можете переглядати контент в Інтернеті, зустрічатися з друзями

і друзями, обмінюватися фотографіями і відео в соціальних мережах. Але однією з основних перспектив цієї ОС є можливість керувати розумним будинком. В даний час існує безліч стартапів і прототипів, а також готові розгортання інтелектуальних домашніх систем, і всі вони повинні управлятися сервером, і найпопулярнішим варіантом клієнтської частини на сьогоднішній день є розгортання на МОС. Тому, на мій погляд, перспективи розвитку всіх мобільних операційних систем в майбутньому дуже великі, а їх тісний зв'язок з інтелектуальними системами будинку просто неможливо обійти.

1.1.2. Що таке XML-файл та чому він важливий у наш час

Файли розширення .xml являє собою текстовий файл у форматі XML – являє собою розширювану мову маркування, яка описує документ і частково описує поведінку програм, що читають документи XML. Формат мови був розроблений з розрахунком його широкого застосування в Інтернеті. Перш за все, він розглядався як заміна мови для маркування HTML, але в результаті він займає своє нинішнє місце. Мова називається розширювальною через можливість вільного розширення маркування шляхом застосування до безпосередніх завдань і потреб використання. В даний час XML широко поширений в Інтернеті, використовується в робочому процесі, на його основі створено безліч додаткових форматів.

Відкрити для перегляду або редагування файлу розширення .xml можна використовувати звичайний текстовий редактор, наприклад блокнот. Ви також можете використовувати Інтернет-браузер, наприклад, Google Chrome, Mozilla Firefox (особливо зручний за допомогою плагіна перегляду XML). Щоб відкрити файл в інтернет-браузері: запустіть браузер, натисніть на клавіатуру Ctrl + O, виберіть потрібний файл XML, натисніть клавішу Enter.

Але це більш правильно, щоб відкрити файл XML в цій програмі або в контексті служби, для якої був створений файл XML.

Специфікація XML описує мову і ряд питань, пов'язаних з кодуванням і обробкою документів. Матеріал цього розділу – це скорочений опис опису мови в специфікації, адаптованої для цієї статті.

Нормативна версія документа вважається англійською, тому основні терміни наведені з їх англійськими оригіналами.

Переклад основних термінів загалом слід доступному в Інтернеті перекладу деталей на російську мову, виключенням є терміни tag і declaration. Для терміна tag використовується переклад тег/тэг. Для терміна declaration краще загальний переклад реклами.

У літературі та інтернеті можна зустріти інші переклади основних характеристик.

Розглядаючи документ з фізичної точки зору, можна зрозуміти, що він складається з об'єктів, з яких кожен може посилатися на інший об'єкт. Єдиним кореневим елементом документа є сутність. Зміст об'єктів – символи.

З іншої (логічної) точки зору документ складається з comments, declaration, посилань на об'єкти та інструкцій з обробки. Все це в документі являє собою структуровану розмітку.

Суть полягає в невеликій частині документа. Всі об'єкти містять щось, і всі вони мають ім'я.

Документ складається з об'єктів, зміст яких символи. При цьому, всі вони розділені на два типи: символічні дані і маркування. До маркування відносяться: tags, що позначають межі елементів, оголошення та інструкції з обробки, включаючи їх атрибути, посилання на об'єкти, коментарі, а також послідовності символів, що обрамляють розділи «CDATA». Частина документа, що не належить до маркування, представляє дані зі знака документа.

Всі складові частини документа підсумовуються в пролозі і кореневому елементі. Кореневий елемент є обов'язковою частиною документа, який становить всю його сутність (пролог, взагалі кажучи, може бути відсутнім). Може включати (і може не включати) вкладені елементи і дані символів, а

також коментарі. Елементи, вкладені в корінь, в свою чергу, можуть включати елементи, вкладені в них, дані символів і коментарі, і так далі. Пролог може включати declaration, інструкції з обробки, коментарі. Він повинен починатися з declaration XML, хоча в певній ситуації допускається відсутність цієї declaration.

Елементи документа повинні бути правильно вкладені: кожен елемент, що входить в інший елемент, повинен бути завершений в елементі, в якому він почався. Дані символу можуть зустрічатися в елементах як безпосередньо, так і в спеціальних розділах «CДАТА». Оголошення, інструкції з обробки та елементи можуть мати пов'язані атрибути. Атрибути використовуються для підключення до логічної одиниці тексту пари ім'я-значення.

Маркування завжди починається з символу «<» і закінчується символом «>». Поряд з символами «<» і «>» особлива роль маркування також грає символ «&». Кутові затискачі означають межі елементів, інструкції з обробки та деякі інші послідовності. Амперсанд дозволяє замінити текст об'єктами.

Використання маркерів в символічних даних ускладнює розпізнавання дизайну маркування і може створити проблему двозначності структури. У XML ця проблема була вирішена таким чином: «<» і «&» не можуть бути присутніми в символ даних і в значеннях атрибутів в їх безпосередній формі для їх представлення в цих випадках були збережені спеціальні сутності (табл. 1.1).

Таблиця 1.1. Відношення спеціальних елементів на їх сутностей

Символ	**Заміна**
<	<
>	>
&	&

Крім того, такі об'єкти використовуються для використання апострофа і лапок в значення атрибутів (табл. 1.2).

Таблиця 1.2. Відношення апострофів,лапок та їх сутності

Символ	**Заміна**
‘	'
“	"

Правило заміни символів, що використовуються в маркуванні, на них позначають об'єкти не застосовуються до символічних даних в розділах «CДАТА», але використовується у всіх інших позиціях документа.

У мові XML всі імена повинні починатися з літери, символу виділення «_» або двокрапки «:» і продовжуватися тільки дійсними для імен символами, в свою чергу вони можуть містити тільки букви, включені в розділ букв кодування Юнікоду, арабські цифри, дефіси, нижня риска, крапка і кома. Однак імена не можуть починатися з рядка XML в кожному реєстрі. Імена, що розпочинаються з цих символів, зареєстровані для використання консорціумом W3C. Слід пам'ятати, що оскільки букви не обмежуються тільки символами ASCII, в іменах можуть використовуватися слова з рідної мови.

XML файл можна розглядати як документ для збереження певної інформації з величезним обсягом. Зручність цього методу полягає в тому, що такий файл легко переноситься з сервера до клієнта і навпаки. XML файл створюється дуже легко, так що ви можете витратити дуже мало часу запису.

Повертаючись до теми розумного будинку, можна уявити собі таку ситуацію: протягом усього тижня в будинку відбуваються різні процеси (споживання енергії, води, різні інші ресурси), які необхідно записати в статистиці. І в кінці тижня ви повинні відновити всю цю інформацію в клієнтську частину користувача. Одним з найпростіших і практичних конверсій буде створення файлу XML і завантаження його з програми на смартфон. Але тоді у нас є завдання, як саме ви можете відкрити цей файл і отримати інформацію, необхідну для відображення на екрані. Для цього використовується парсинг, або іншими словами – семантичний аналіз.

1.1.3. Що таке семантичний аналіз або парсинг XML-файлів

Одним з найважливіших компонентів технології XML є особливий клас програм, що розроблений для аналізу документів і відгук необхідної інформації – парсери. Саме про них ми і поговоримо. Давайте подивимося, для чого потрібні парсери, які вони є.

XML документ зберігає інформацію не в масиві, а у вигляді ієрархічно пов'язаних фрагментів. Оскільки текстові документи дуже легко створювати і відправляти по мережі, вони є надзвичайно вдалим способом зберігання інформації і часто використовуються при розробці складних розподілених додатків (рис. 1.1).

Рисунок 1.1 – Розподіл даних в додатку

Але універсальність формату тексту XML стає досить очевидною неприємністю – перш ніж витягувати дані з документа, потрібно мучити себе синтаксичним аналізом тексту і визначити його структуру. Виконання всіх необхідних процедур вручну – досить нетривіальна діяльність і вимагає значних зусиль. Одним із стандартних механізмів для спрощення життя розробників і є парсери.

XML аналізатор являє собою програму, призначену для аналізу вмісту текстового документа, який відповідає специфікації XML. Йде вся «чорна» робота: отримання загальної інформації про документ, аналіз тексту, Пошук в

ньому складних конструкцій (елементи, атрибути, об’єкти і т.д.) для перевірки дотримання синтаксичних правил, а також забезпечити інтерфейс для доступу до документа. В результаті ретельно витягнуті дані будуть передані користувачеві з програми, яка може нічого не знати про те, що таке XML.

Парсер може бути виконаний як окремий програмний модуль або компонент активації, може під’єднуватися до додатка через особливі бібліотеки класів на етапі компіляції або run-time виконання. Парсери діляться на неверифікуючі та верифікуючі. Одні можуть перевірити структуру документа на основі схем даних, а другі не зацікавлені в цьому – і тому зазвичай мають менший розмір. Багато сучасних парсерів «завантажені» безліччю додаткових функцій (розширене пропрацювання помилок, додавання і зміна даних), що відображає їх, як більш зручні для роботи, хоча і збільшує розмір програм. Майже всі загальні парсери також підтримують ряд важливих стандартів XML – або поставляються разом з парсером на інших мовах, отриманих з нього.

Найбільш поширеним і відомим є парсер Експат (Expat), написаний Джеймсом Кларком – одним з творців специфікації ЦМЛ. Він працює на мові програмування C++ і поширюється разом з open-source кодом. До слова, саппорт цієї мови маркування в таких відомих середовищах, як ПХП і Перл здійснюється на його основі. Іншим загальним парсером – Ксеркес (Xerces) є проект Апаче XML (реалізований на мовах Java і C ++). Ви можете знайти багато парсерів для C ++, Perl і Python. Велика частина їх написана на Java і підходить для будь-якої платформи, відомої на Java. Лідери ринку (Майкрософт, Оракл) завжди мають масштаб і монументальність. Вони випустили більш важкі та функціональні пакети, які вони містять, на додаток до фактичних парсерів, безліч додаткових утиліт, які полегшують життя розробникам.

На платформі Андроїд є також локальні інструменти для завантаження файлів на основі XML на основі Джава і Android SDK. Серед них:

XMLPullParser, DOM Parser, SAX Parser. Вони будуть обговорюватися в наступному підрозділі.

У цій дисертації об'єктом дослідження є семантичний аналіз XML-файлу на мобільній платформі Андроїд. Тема дуже популярна в даний час і перспективна для розвитку в майбутньому.

1.2. Виділення проблеми та визначення предмета дослідження

Є багато бібліотек для обробки файлів XML на платформі Андроїд і мовою програмування Java, але три найбільш популярних серед них були обрані для огляду: DOMParser , XMLPullParser, SAXParser.

1.2.1. Огляд XMLPullParser

XmlPullParser – XML є аналізатором, який може бути використаний для аналізу документа XML. Принцип його роботи полягає в тому, що він проходить через весь документ, зупиняючись на його елементах. Але він працює не поодинці, а за допомогою методу «далі». Ми постійно закликаємо до методу «далі», і за допомогою методу «getEventType» ми перевіряємо, який елемент зупинив алгоритм.

Основні елементи документа, які захоплює парсер:

- START_DOCUMENT – старт документа;
- START_TAG – старт тега;
- TEXT – склад елемента;
- END_TAG – закінчення тега;
- END_DOCUMENT – закінчення документа.

Існують різні типи парсингу залежно від того, які прапорці встановлено:

- неприпустимий аналізатор, який визначається в специфікації XML 1.0, якщо параметр FEATURE_PROCESS_DOCDECL налаштований на true;

- припустимий аналізатор, коли FEATURE_VALIDATION встановлено на true (і це означає, що FEATURE_PROCESS_DOCDECL встановлено на true);
- коли FEATURE_PROCESS_DOCDECL становлено на false (це за замовчуванням, і якщо вам потрібно інше значення має бути змінено, перш ніж запускати синтаксичний аналіз), то парсер поводиться як невідповідний аналізатор, сумісний з XML 1.0 (визначається з функцією *defineEntityReplacementText*()). Такий режим парсингу призначений для роботи в певних обмежених середовищах.

Існує два ключові методи: *next*() і *nextToken*(). Хоча *next*() забезпечує доступ до подій синтаксичного аналізу високого рівня, *nextToken*() забезпечує доступ до маркерів нижчого рівня.

Поточний стан події синтаксичного аналізатора може бути визначений шляхом запуску методу *getEventType*(). По-перше, синтаксичний аналізатор знаходиться в стані START_DOCUMENT.

Next() призводить аналізатор до наступної події. Значення введення, яке повертається з наступного, визначає поточний стан парсера і ідентичний значенню, яке повертається з наступних *getEventType()* викликів.

Після першого *next*() або *nextToken*() (або іншого *next*() методу) додаток може отримати версію XML, автономність і кодування з декларацією XML наступними способами:

- версія: *getProperty("http://xmlpull.org/v1/doc/properties.html#xmldecl-version")* повертає *String("1.0")* або null, якщо документ не був прочитаний або якщо властивість не підтримується;
- автономність: *getProperty("http://xmlpull.org/v1/doc/properties.html#xmldecl-standalone")* повертає логічне значення: null, якщо не було окремої декларації або якщо властивість не підтримується, інакше п.овертається логічне значення true, якщо автономний режим присутній, в інакшому випадку false;

- кодування: отримано від *getInputEncoding()* null, якщо потік мав невідоме кодування (не встановлено в *setInputStream*).

Цей інструмент дуже популярний в розробці андроїд додатків, але не відрізняється здатністю швидко обробляти документ, оскільки він переміщається по всьому файлу за один раз, кожен раз, коли необхідно щось знайти, цей процес буде повторюватися.

1.2.2. Огляд парсеру DOMParser

DOM parser використовує об'єктний підхід при створенні та аналізі файлів XML в андроїд додатках.

Як правило, **DOM parser** завантажує файл XML в пам'ять для аналізу документа XML, оскільки він буде споживати більше пам'яті і аналізувати документ XML від вихідного вузла до кінцевого вузла.

В принципі, файл XML буде містити 4 основних компоненти (табл. 1.3).

Таблиця 1.3. Складові DOM parser

Компонент	**Опис**
Пролог	Як правило, файл XML почнеться з прологу. Перший рядок, що містить інформацію про файл – пролог.
Події	Файл XML буде містити багато подій, які включають початок і кінець документа, початок і закінчення тегів і т.д.
Текст	Це простий текст в елементах тегів xml.

Закінчення таблиці 1.3

Атрибути	Атрибути – це додаткові властивості тега, такі як значення і тд, які знаходяться в тегах.

Щоб прочитати і проаналізувати дані з XML з використанням DOM парсера в андроїд, нам необхідно створити копію спеціальних об'єктів в андроїдному додатку, які використовуються для аналізу даних XML для отримання необхідної інформації.

Істотною перевагою над іншими XML-парсерами є те, що цей інструмент все ще здатний створювати файли XML.

Істотним недоліком цього інструменту є той факт, що DOM парсер завантажує весь файл XML замість пам'яті, що значно уповільнює обробку, а також дуже сильно завантажує систему.

1.2.3. Огляд SAXParser

В андроїд SAX виступає в якості простого апі для XML і широко використовується для аналізу XML.

Основною перевагою синтаксичного аналізу Сакса над DOM парсером є те, що ми можемо доручити Сакс парсеру зупинитися в середині документа, не втрачаючи вже зібраних даних.

Як і DOM парсер, Сакс парсер також використовується для виконання операцій в пам'яті для розбору документа XML, але споживає менше пам'яті в порівнянні з DOM парсером.

Sax аналітик перевіряє XML файл, символ в символ і переводить його в ряд подій, як startElement(), endElement() і characters(). Об'єкт контент-менеджер буде обробляти ці події для реалізації відповідних кроків, а метод parse() відсилає події на об'єкт контенту, щоб співпрацювати з ним.

Для того, щоб читати і аналізувати дані про XML за допомогою синтаксичного аналізатора Сакс в андроїд – необхідно створити копію об'єкта

SAXParserFactory в андроїд додатоку, який з серією подій, для того, щоб отримати необхідну інформацію з об'єктами XML.

Як вже можна зрозуміти, цей інструмент також не може обробляти великі за обсягом документи протягом короткого часу, так як він також завантажує вміст файлу XML в пам'ять гаджета.

1.2.4. Швидкість обробки великого обсягу даних – одна з найголовніших проблем при парсингу файлів

Повернемося до нашого прикладу за допомогою програми для управління системами інтелектуального будинку. Протягом усього тижня сервер збирає інформацію і хоче відправити її нам на мобільний девайс (клієнт) під виглядом файлу XML.

Будь-яка компанія, яка замовила продукт, програмне забезпечення, в нашому випадку це клієнтський додаток на мобільній осі, завжди ставить такі умови, що користувачеві приємно їх використовувати, а продукт, в свою чергу, повинен бути зрозумілим і легко опановуваним.

Тому програма не повинна бути заторможеною, завжди повинна бути обробка всіх можливих помилок і виводити їх в користувальницький інтерфейс. Також не повинно бути довгих завантажень, які перекривають користувачевы зв’язок з візуальним інтерфейсом програми. У нашому випадку обробка великого обсягу файлу XML може зайняти багато часу і навіть блокувати основний користувальницький інтерфейс. Ось чому вищезгадані інструменти можуть не підходити для застосування цього типу. Тому виникає питання, як саме ви можете обробляти вміст файлу XML швидше і продуктивніше для системи гаджета.

1.2.5. Багатопоточне програмування – один із інструментів для швидкої обробки даних

Після того, як інженери придумали багатопроцесорні системи, перед програмістами відразу виникла проблема: чи можливо з найменшими

витратами створити паралельне виконання коду. Хоч з тих пір пройшло багато часу, основні принципи не змінилися.

Процес являє собою набір коду і даних, які поділяють ВАП. Часто програма вміщує в собі один процес, але існують винятки (наприклад, браузер Хром створює окремий процес для кожного розділу, що дає йому деякі переваги, такі як незалежність розділи один від одного). Процеси роботи відділяються один від одного, тому доступ до ресурсів іноземного процесу нездійсненний (зв'язок між процесами створюється за допомогою особливих засобів).

Один потік є одиницею для виконання коду. Кожна нитка послідовно виконує вказівки процесу, до якого вона належить, поряд з іншими потоками цього Лінукс процесу.

Щодо фрази «паралельно з іншими потоками». Відомо, що ядро процесора в кожній часовій точці має один блок перетворення. Це означає, що одноядерний процесор може обробляти команди тільки послідовно, один за іншим (спрощеним способом). Однак запуск декількох паралельних потоків також можливий в одноядерних системах процесорів.

У цьому випадку система періодично перемикається між потоками, послідовно даючи один або інший потік, який буде виконаний. Така схема називається псевдопараллізмом. Система запам'ятовує стан кожного потоку, перш ніж переходити до іншого потоку, і відновлює його, як тільки він повертається до виконання потоку. Значення потоку містить параметри, такі як стек, адреса виконуваних файлів і багато іншого ...

Як вже можна зрозуміти, застосування семантичного аналізу файлу XML за допомогою багатопотокових інструментів підходить для нашої проблеми. Таким чином, можна буде домогтися хороших результатів в економії часу на обробку документа.

Таким чином, предметом дослідження в цій дисертації є методи обробки вмісту файлів XML за допомогою багатопоточного програмування.

1.3. Постановка задачі

Створено завдання: визначити інструменти і методи, які допомагають працювати з багатопоточністю в Java (мові програмування) і в ОС Андроїд. Потім потрібно створити програму, яка може показати, як живий приклад, час повної обробки великого обсягу файлу XML з використанням різних методів, що застосовуються в цьому додатку. Потім треба виконати систематичні тести і порівняти показники, після чого вже буде можливо створити унікальний алгоритм для виявлення найоптимальнішого метода при певних умовах.

Висновок до розділу

У цьому розділі розглядаються те, що таке файл XML, кк правильно його використовувати і як його обробляти. Визначено предмет дослідження для, об'єкт дослідження, проблема, і можливі завдання. Також пропонується сценарій, в якому в майбутньому розробникам додатків для різних МОС доведеться вирішувати ці проблеми.

2. АНАЛІЗ ПРОБЛЕМ ШВИДКОГО СЕМАНТИЧНОГО АНАЛІЗУ ВЕЛИКОГО ОБСЯГУ ДАНИХ

2.1. Багатопоточність

Багатопоточність має бути реалізована за допомогою різних моделей і різних сценаріїв. Кожен з них відрізняється швидкістю, навантаженням на систему і потребами у використанні.

2.1.1. Огляд поняття багатопоточності

Синхронна модель програмного забезпечення – це модель програмного забезпечення, коли одне завдання призначено потоку, і починається реалізація. Коли завдання завершено, тобто можливість зайнятися іншим завданням. У цій моделі неможливо зупинити виконання завдання, щоб виконати інше завдання між ними. Давайте обговоримо, як ця модель працює в одному і багатопотоковому сценарії.

Однопоточність – якщо у нас є кілька завдань, які повинні бути виконані, і поточна система забезпечує один потік, який працює з усіма задачами, він приймає один за іншим, і процес виглядає як на рис. 2.1.

Рисунок 2.1 – Сценарій однопоточності

Тут ми бачимо, що у нас є потік (потік 1) і 4 завдання, які повинні бути виконані. Потік починає виконувати послідовно один за іншим і виконувати їх усі. Порядок виконання завдань не впливає на загальну продуктивність, у нас може існувати інший алгоритм, який визначає пріоритети завдань.

Багатопоточність – у цьому сценарії ми використовували багато потоків, які можуть виконувати завдання і почати працювати з ними. У нас є потоки потоків (нові потоки також створюються на основі потреб і

доступності ресурсів) і багатьох завдань. Отже, потік може працювати так, як показано на рис. 2.2.

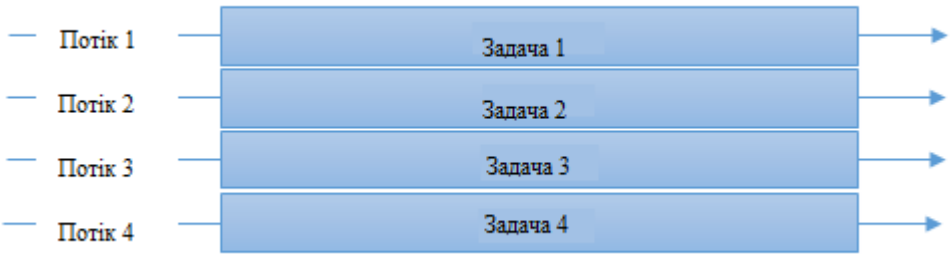


Рисунок 2.2 – Сценарій багатопоточності

Можна побачити, що є 4 потоки і стільки ж задач для виконання, і кожен потік працює з ними. Це майже ідеальний сценарій, але за нормальних обставин ми використовуємо більшу кількість завдань, ніж кількість доступних потоків, тому випущений потік отримує інше завдання. Як уже згадувалося, створення нового потоку відбувається не кожен раз, тому що для визначення системних ресурсів, таких як процесор, пам'ять і вихідна кількість потоків.

Тепер розглянемо асинхронні моделі і про те, як він поводиться в одно і багатопотоковому середовищі програмування.

Асинхронна модель програмування – відрязняється від синхронної моделі програмування тим, що тут потік, як тільки ви починаєте виконання завдання, може зупинити виконання, збереження поточного стану і тим часом почати виконання іншого завдання(рис . 2.3).

Рисунок 2.3 – Сценарій асинхронної однопоточної моделі

Можна побачити, що один потік виконує все завдання, а завдання чергуються один за іншим.

Якщо наша система може мати багато потоків, то всі потоки мають працювати по асинхронній схемі, як показано на рис. 2.4.

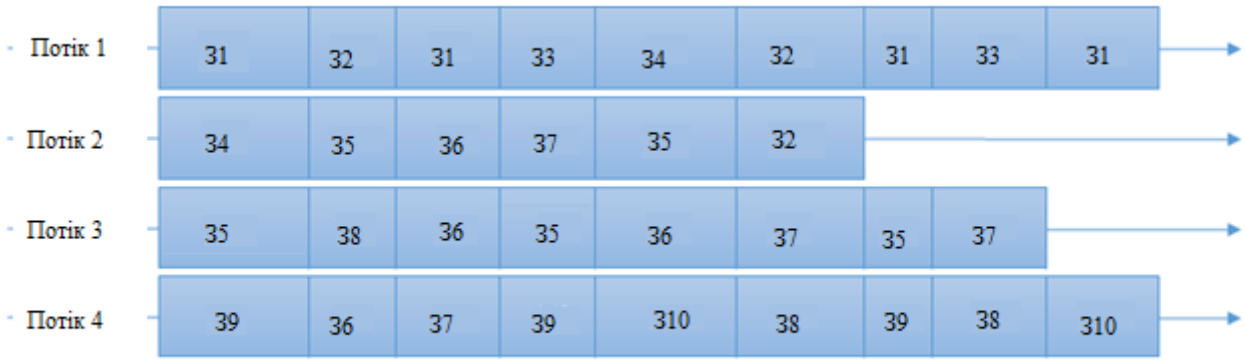


Рисунок 2.4 – Сценарій асинхронної багатопоточності

Тут ми можемо побачити те ж завдання, наприклад, T4, T5, T6 ... вони обробляються декількома потоками. Це краса цього сценарію. Як ми бачимо, завдання T4 було виконано першим потоком 1 і завершено потоком 2. Точно так же завдання T6 виконується потоком 2, потоком 3 і потоком 4. Це вказує на максимальне використання потоків.

Отже, до сих пір ми обговорили 4 сценарії:

- Синхронний однопотоковий;
- Синхронний багатопотоковий;
- Асинхронний однопотоковий;
- Асинхронний багатопотоковий.

Розглянемо ще один термін – паралелізм.

Паралелізм є способом обробки декількох запитів одночасно. Оскільки ми обговорювали два сценарії при обробці декількох запитів, багатопотокового програмування і асинхронної моделі (одиночної і багатопотокової). У випадку асинхронної моделі, будь то одиночна або

багатопотокова, в той час, коли виконується безліч завдань, деякі зупиняються, а деякі виконуються.

Як обговорювалося раніше, нова ера – це ера асинхронного програмування. Чому це так важливо?

Є дві речі, які дуже важливі для кожної програми – зручність і продуктивність. Юзабіліті, тому що користувач натискає кнопку, щоб зберегти деякі дані, які, в свою чергу, вимагає виконання багатьох задач, таких як зчитування і заповнення даних у внутрішньому приклады, встановлення зв'язку з СКЛ і збереження його там.

У свою чергу, СКЛ запускається на другій машині в мережі і відповідає під іншим процесом, це забирає багато часу. Отже, якщо запит опрацьовується одним процесом, а екран буде застоюватися до завершення процесу. Ось чому сьогодні багато програм повністю покладаються на асинхронну схему.

Продуктивності мобільних додатків і системи також важливі! Він контролюється під час виконання запиту, близько 70-80% з них потрапляють в очікуванні залежних завдань. Таким чином, він також може бути максимально використаний в асинхронному програмуванні, де, як тільки передача передається іншому потоку (наприклад, СКЛ), поточний потік зберігає стан, а також стає доступний для виконання іншої задачі, і коли завдання СКЛ закінчується, будь-який вільний потік може зробити цю роботу.

2.1.2. Паралельне програмування

Паралельна програма – це дуже Взаємодіючі паралельні процеси. Основна мета паралельних обчислень – прискорити вирішення обчислювальних проблем. Паралельні програми мають такі особливості:

- здійснюється управління роботою безлічі процесів;
- організовується обмін даними між процесами;
- втрачається детермінізм поведінки через асинхронності доступу до даних;
- переважають нелокальні та динамічні помилки;

- з'являється можливість тупикових ситуацій;
- виникають проблеми масштабованості програми і балансування завантаження обчислювальних вузлів.

Подумайте про якийсь послідовний алгоритм для вирішення кожної проблеми. У ньому існують операції, які не можуть працювати паралельно (наприклад, введення / виведення), так і операції, які можуть виконуватися на двух або більше процесорах одночасно. Припустимо, що частка послідовних операцій дорівнює a.

Час виконання послідовного алгоритму означає T1. Час виконання паралельної версії алгоритму P ідентичних процесорів можна записати наступним чином:

$$S = \frac{T_1}{T_p} = \frac{T_1}{\alpha * T_1 + \frac{(1 - \alpha) * T_1}{p}} \leq \frac{1}{\alpha} \qquad (2.1)$$

Прискорення паралельного алгоритму називається співвідношенням часу виконання кращих послідовних алгоритмів до часу виконання паралельного алгоритму:

$$S = \frac{T_1}{T_p} \qquad (2.2)$$

Паралельний алгоритм може дати велике прискорення, але для цього неефективно використовувати безліч процесів. Концепція ефективності використовується для оцінки масштабованості паралельного алгоритму:

$$E = \frac{S}{p} \qquad (2.3.)$$

Теоретична оцінка максимального прискорення, досяжна для паралельного алгоритму з пропорцією послідовних операцій, рівною α, визначається законом Амдала:

$$S = \frac{T_1}{T_p} = \frac{T_1}{\alpha * T_1 + \frac{(1 - \alpha) * T_1}{p}} \leq \frac{1}{\alpha} \qquad (2.4)$$

Таким чином, якщо тільки 10% операцій алгоритму не можуть виконуватися паралельно, то паралельне виконання цього алгоритму може дати прискорення в 10 разів або більше.

2.2. Огляд вже існуючих методів на Java та у Android SDK для роботи з багатопоточністю

Є три основні бібліотеки, які допомагають працювати з потоками при створенні додатків для МОС Андроїд.

2.2.1. Огляд Threads and Locks

Якщо ваша система має тільки один процесор, потоки виконуються в свою чергу (але швидке перемикання системи між ними імітує паралельну або одночасну роботу). Рис. 2.5 показує додаток, який має три потоки продуктивності:

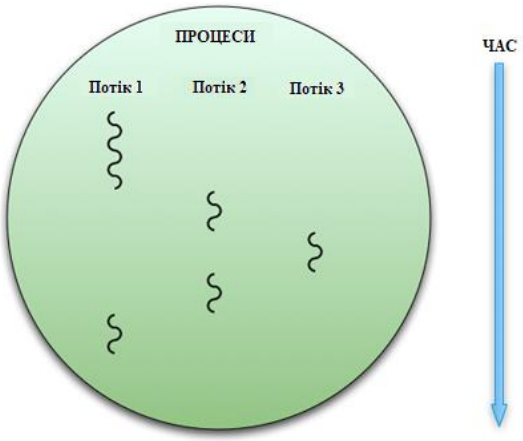


Рисунок 2.5 – Виконання 3-х потоків, схема

Потоки можуть виконувати декілька задач одночасно, не зважаючи один на одного, що дає можливість ефективно використовувати ресурси системи. Потоки можуть бути використані тоді, коли одна тривала дія не повинна заважати іншим діям. Наприклад, у нас є музичний плеєр з кнопками відтворення і паузи. Якщо ви натиснете кнопку відтворення та запустите музичний файл в окремому потоці, ви не можете натиснути кнопку паузи, поки

файл не буде повністю відтворюватися. За допомогою потоків можна обійти ці проблеми.

2.2.2. Огляд RxJava

До того, як ми почнемо обговорювати механізми RxJava, давайте згадаємо модель спостерігача. У ньому є об'єкт, який генерує будь-які події і має об'єкт або об'єкти, які реєструються і отримують ці події. Я думаю, що на роботі ви постійно використовуєте цю модель. Найпростішим прикладом є маніпулятор з натисканням кнопки.

Ява навіть має інструменти для цієї моделі-класу спостережуваний і інтерфейс спостерігача. Реалізація інтерфейсу спостерігача – це об'єкт, який очікує події. Спостережуваний – це клас, який з усіх об'єктів, які передали йому спостерігач, повідомить, що подія відбулася.

Ті ж Імена також використовуються в RxJava. І їх значення залишається незмінним: спостережуваний генерує подію і спостерігач отримує його. Але саме поняття «подія» було значно розширено. У RxJava подій, які Observable передає Observer, може розглядатися як потік даних. Задачі в цьому потоці мають три можливі типи:

- Next – чергова порція даних;
- Error – сталася помилка;
- Completed – потік завершений і даних більше не буде.

2.2.3. Огляд Kotlin Coroutines

Коротіни спрощують асинхронне програмування, приховуючи всю складність в бібліотеках.

Ця ідея виникла в далекому 1967 році у програмній мові Симула. Спочатку було всього кілька методів: *detach* і *resume* (тому вони дозволили зупинити і запустити виконання).

Однак ідея була забута тоді, коли з'явилися потоки. Хоча має сенс виконати асинхронну задачу в окремому потоці. Однак у деколи це рішення не буде достатньо ефективним.

Розробники говорять про coroutines таке: «coroutines є розрахунками, які можуть бути зупинені, не блокуючи потік».

«Не блокувати потік», що це означає? Простіше кажучи, мова йде про функцію, яка може бути запущена, відокремлена і відновлена з одного і того ж місця. Короутіни – це новий, зручний спосіб виконання неблокуючих асинхронних операцій. Крім того, створення коротинів – це простіша операція в порівнянні зі створенням потоків.

2.3. Деталізація постановки задачі

Таким чином, аналізуючи існуючі бібліотеки для роботи з багатопотоковою роботою та аналізуючи концепції паралельного програмування та асинхронності, детально визначаємо задачу.

1. В результаті аналізу бібліотек для роботи з багатопоточністю, а саме RxJava, Kotlin Coroutines, Threads, визначити перелік операцій, які реалізують паралельне та багатопоточне програмування.

2. Запропонувати склад уніфікованої бібліотеки, яка синтезує всі основні методи обробки данних XML-файлу.

3. Розробити власну бібліотеку в якій реалізувано метод сбору контенту з XML-файла.

4. Провести апробацію запропонованої бібліотеки розробивши прикладний додаток.

5. Провести аналіз усіх методів, після чого розробити унікальний алгоритм для визначення найоптимальнішого метода при певних обставинах.

Висновок до розділу

Були описані передові бібліотеки для багатопотокової роботи, серед яких RxJava, Threads, Kotlin Coroutines. Також розглядаються такі поняття, як

багатопоточність, асинхронність і паралельність. В результаті було детальне визначення завдання за випускним проектом.

3. РОЗРОБКА ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТА

3.1. Задач та підзадачі, що вирішуються в додатку, а також їх склад та ієрархія

Потрібно створити мобільний додаток для платформи андроїд. Додаток повинен містити приклади семантичного аналізу з різними методами і інструментами. Для створення власної Android програми потрібно добре розуміти процес запуску додатку на девайсі (див. Додаток В). Наприклад, два файли XML беруться з зовнішнього носія, який також впливає на час обробки даних. Користувач вибирає кожен з методів і спостерігає на дисплеї, як саме він впливає на основний потік програми, і в кінці він отримує результат обробки (час, протягом якого алгоритм повністю управляється нашими файлами).

Перш за все, необхідно створити необхідну кількість алгоритмів з використанням різних бібліотек, інструментів. Серед них будуть: Thread and Locks, RxJava і Kotlin Coroutines. Кожен з методів в кінці, повинен дати унікальний результат час обробки, який з кожним запуском повинен бути трохи відрізняється, так як XML файли для великих, і система може впливати на будь-який зовнішній фактор (послуги інших програм, стандартні програми, які не можуть викинути з пам'яті).

Наступна глобальна задача – створити інтерфейси програми, які дозволять користувачеві швидко і без будь-яких проблем зрозуміти, де знаходиться алгоритм, побачити результат цього методу.

Для вирішення завдань використовується наступна ієрархія.

Перед користувачем з'являється меню з розробленими мною алгоритмами, Користувач настискає на кожному з них, система здається, у вас є дозвіл на читання і з одного зовнішнього носія, якщо не хоче, якщо це так, читає два XML файлів і починає працювати з ними. На основному потоці ми отримуємо вікно завантаження, яке блокує основний потік, в кінці алгоритму вікно

завантаження зникає, і користувач отримує повідомлення з результатами виконання алгоритму.

3.2. Вхідні та вихідні інформаційні потоки для кожної задачі та підзадачі

Система управління приймає повідомлення після того, як користувач вибирає конкретний метод обробки.

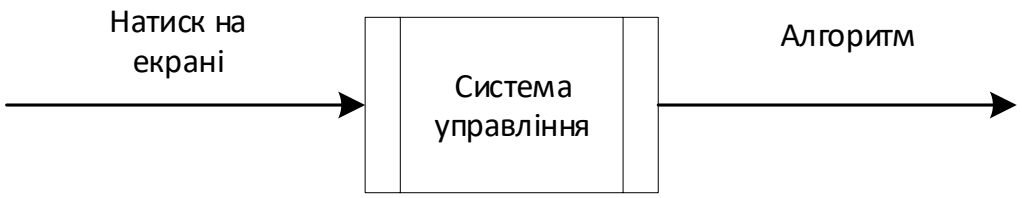


Рисунок 3.1 – Потоки процесу СУ, інформаційні

Система процесу: приймає повідомлення від девайса про наявність можливості роботи з зовнішнією пам’яттю і отримує файли саме з неї.

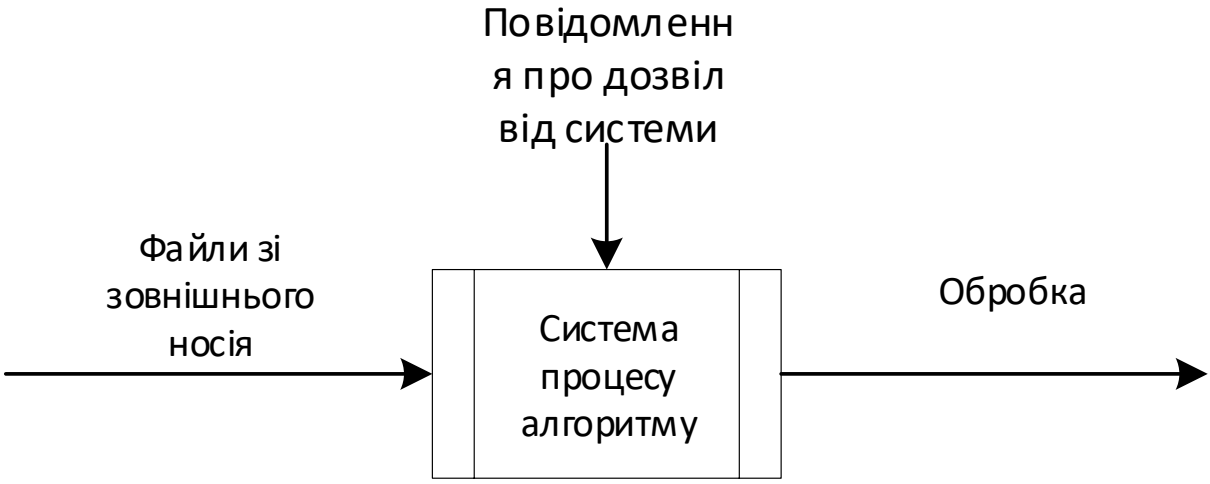


Рисунок 3.2 – Потоки процесу оновлення, інформаційні

3.3. Послідовність вирішення задач та підзадач

По-першу, необхідно створити алгоритми парсингу даних, так як без них працювати далі неможливо. По-друге, як раніше було визначено, скільки

методів обробки у нас буде і які саме, а також їх застосування, необхідно розробити простий інтерфейс, за допомогою якого людині буде легко зрозуміти, де з методів знаходиться і побачити результати його виконання.

Ми також не повинні забувати, що ми повинні блокувати основний потік, для якого розробляється діалогове меню завантаження, яке не може бути відключено до тих пір, поки обробка не буде завершена остаточно. І, звичайно ж, користувач повинен показати результати обробки (час).

Використовуючи все вищесказане, кожен користувач може вибрати найбільш підходящий для себе спосіб обробки, який йому найбільше підходить.

3.4. Алгоритм функціонування додатку

Після запуску програми користувач бачить перед собою привітання і красиво розроблену єдину кнопку, яка кидає його на актівіті методів обробки.

Після вибору будь-якого з методів на екрані вибору методів Користувач контролює діалог завантаження, який блокує основний потік і результати після завершення алгоритму (див. Додаток Г).

3.5. Алгоритми вирішення задач

Додаток включає в себе 7 методи обробки XML-файл за допомогою 3х інструменти: Threads, RxJava, Kotlin Coroutines. У кожного з інструментів існує два методи (парсинг двох файлів в одному або декількох потоках). Останній інструмент також включає обробку в чотирьох нитках, що значно підвищує ефективність алгоритму. RxJava також має можливість паралельної обробки методом *parallelStream*(). Для детального огляду див. Додаток Г.

Кожен метод здійснюється через один алгоритм, але різні інструменти і методи, які надає нам певну бібліотеку. Виділяється необхідна кількість потоків, що працюють з даними, взятими з файлу XML, який слід розглядати як початок (для детального розуміння алгоритму парсингу див. Додаток Ґ).

Для того, щоб додаток дійшло до нашого файлу XML, система повинна повідомити його про те, що у нього є дозвіл на роботу з зовнішніми носіями, в іншому випадку буде виведення, за допомогою якого додаток автоматично закриється. Таким чином, на головному екрані при натисканні кнопки користувачеві буде запропоновано дозвіл на доступ до зовнішнього носія, якщо Користувач відмовиться, то нічого не станеться далі, якщо він погодиться, це з'явиться вікно вибору методу обробки даних.

Дозвіл запитується тільки один раз у разі позитивної відповіді, якщо користувач відмовляється надати доступ, система попросить його знову, коли ви натиснете одну клавішу в головному меню.

3.6. Склад головного меню

Головне меню досить просте і складається з одного елемента – кнопки, яка переводить користувача в меню вибору методів обробки.

Меню методів обробки складається з 7 кнопок (кожен відповідає за окремий алгоритм), а також опис кожного з алгоритмами.

3.7. XML файли для прикладу

На зовнішньому носії має бути два файли XML, кожен з яких важить 28 і 24 мегабайт відповідно. Ці файли були завантажені з ресурсу Вікіпедії з обох його сторінок.

Ці файли дуже великі і будуть добре відображати різницю між алгоритмами, наведеними в додатку.

В цілому ці два файли мають 196681 слова, які відображаються кожного разу після закінчення процесу обробки (кожен з алгоритмів відноситься до загальної кількості слів).

3.8. Запити до системи для отримання дозволів

Гугл дуже турбується про безпеку своєї операційної системи, так що ввести таку річ, як дозволу виконання. Є деякі дозволи, які можна побачити при

установці програми з GooglePlay, і є ті, які будуть запитувати в самому процесі використання програми.

Якщо ви встановите запит на дозволи у формі, яка не ясна для користувача, він може не підтвердити їх. В результаті юзер не отримає можливості працювати з додатком, а інколи навіть втратить довіру до софта і видалить його з девайса.

У нашому випадку користувач повинен запросити такі дозволи, як WRITE_EXTERNAL_STORAGE і READ_EXTERNAL_STORAGE, інші в додатку не зможуть читати дані, а саме XML файли з зовнішнього середовища пам’яті.

Процес запиту пермішина можна побачити на рис. 3.3 і рис. 3.4.

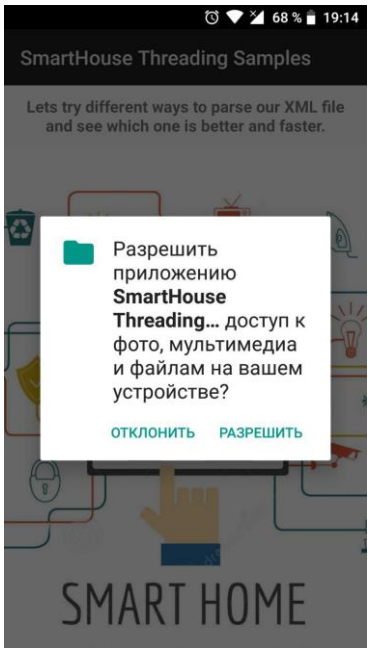


Рисунок 3.3 – Запит системи для користувача щодо доступу до Додатка до зовнішнього носія

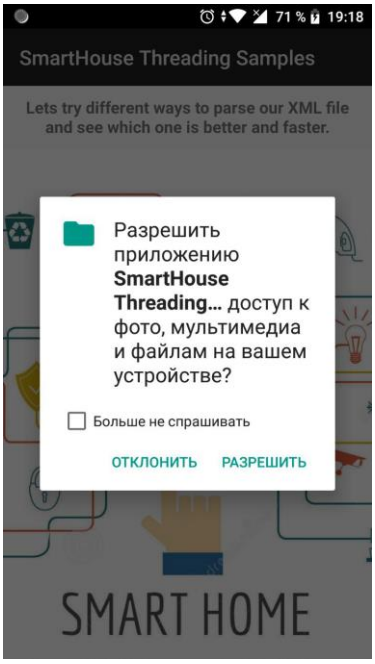


Рисунок 3.4 – Запит системи користувачеві про доступ додатка до зовнішнього носія в разі, якщо він вперше був відмовлений в доступі

3.9. Запити до зовнішнього середовища пам’яті для подальшого отримання файлів

Оскільки розміщення двох файлів XML в додатку не було б дуже розумним (розмір файлу АПК збільшився б на 50 мегабайт), було вирішено перемістити ці файли вручну в конкретний каталог.

За замовчуванням цей каталог вказаний в коді статичними змінними в класі *Source*:

```
private fun wikiPages(fileName: String) =
File(Environment.getExternalStorage.Directory().path + FILES_DIRECTORY +
fileName)
private val FILES_DIRECTORY = "/Diploma/"
private val FILE_NAME_WIKI_PAGES_01 = "wiki_pages_big_01.xml"
private val FILE_NAME_WIKI_PAGES_02 = "wiki_pages_big_02.xml"
```

Також створено два методи, які відповідають за повернення об’єктів класу *File*, тобто XML-файлів, зчитаних зі зовнішнього носія:

```
fun wikiPagesBatchOne() = wikiPages(FILE_NAME_WIKI_PAGES_01)
fun wikiPagesBatchTwo() = wikiPages(FILE_NAME_WIKI_PAGES_02)
```

Щоб прочитати весь вміст файлу ММЛ, розроблений алгоритм. На самому початку створюється об'єкт класу *Pages*, який успадкований від ітераційного інтерфейсу. У параметрах конструктора передається вже раніше прочитаний файл із зовнішнього носія. У середині цього класу був створений інший клас *PageIterator*, який успадкований від інтерфейсу ітератора. Цей клас відповідає за читання вмісту файлу, список всіх символів, перехід за тегами і вимірювання часу.

Також були створені класи *Words* (ітераційний) і внутрішній клас *WordIterator* (ітератор), які відповідають за читання вмісту сторінки файлу XML. Докладний програмний код всіх створених класів для роботи з XML файлами і їх зміст можна побачити в додатку Б.

3.10. Звіт про завершення парсингу файлів

Після завершення парсингу юзер отримує звіт про результат обробки (час і кількість слів, присутніх в цілому в обох документах)

Такий звіт може бути отриманий після будь-якого із робочих запитів.

Приклад такого результату можна спостерігати на рис. 3.5.

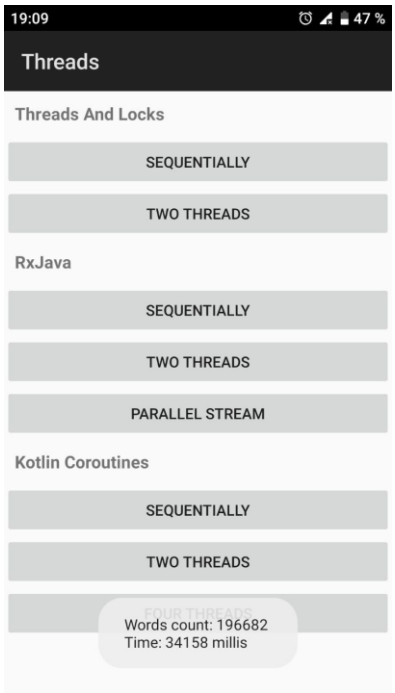


Рисунок 3.5 – Результат роботи додатку після завершення обробки

3.11. Форми меню

Додаток складається з двох екранів.

Рисунок 3.6 – Екран привітання

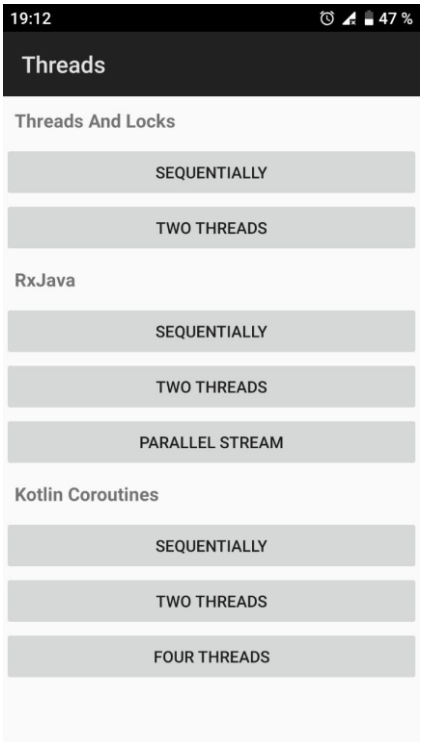


Рисунок 3.7 – Екран методів обробки

Як ми бачимо, на першому екрані меню складається тільки одна кнопка, яка являє собою зображення, на другому екрані є 8 кнопок, кожна з яких відповідає певному методу обробки.

3.12. Принципи проектування інтерфейсу користувача

Додаток використовує комбінацію принципів для розробки графічного інтерфейсу користувача та меню інтерфейсу.

Графічний інтерфейс користувача (графічний інтерфейс користувача) – тип інтерфейсу, в якому усі деталі інтерфейсу, які можна побачити на екрані, розроблені у вигляді інтерактивних графічних зображень. В графічному інтерфейсі (ГІК) надається довільний доступ (через пристрої введення) до усіх існуючих екранних об'єктів (або елементів інтерфейсу) і прямих маніпуляцій з ними. Найчастіше елементи інтерфейсу в ГІК виконуються на основі метафор і відображають їх цілі і властивості, що полегшує розуміння і поглинання недосвідченими користувачами.

Меню – користувальницький інтерфейс, який представляє собою список можливих дій, що відображаються на екрані або у вікні для користувачів, щоб вибрати потрібні параметри. Меню має бути важливою частиною ІК, вони дають можливість користувачам переміщатися по програмному забезпеченню, надаючи можливість вибору бажаних елементів або опцій для виконання операцій. Меню різні за формою, розміром і стилем. У повноекранному меню доступні функції або завдання представлені у вигляді списків на екрані. Визначальною особливістю більшості повноекранних меню є їх ієрархічна структура. При виконанні завдання користувачі повинні переміщатися по структурі дерева меню.

Рис. 3.8 і фіг. 3.9 ви можете побачити схематичне зображення екранів, які доступні в нашому додатку:

Рисунок 3.8 – Схема екрану привітання

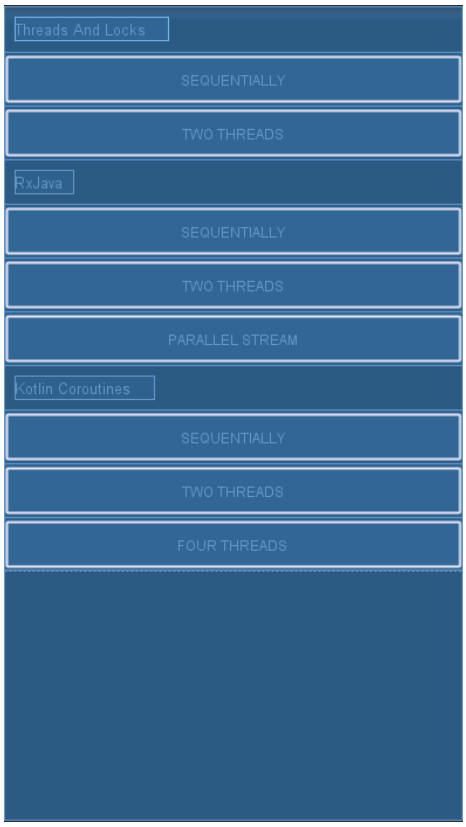


Рисунок 3.9 – Схема екрану методів обробки

3.13. Структура комплексу програми

Каталог створеної програми містить файли (UML-діаграму див. у додатку Д):

- 2 *Activity:*
    - *MainActivity;*
    - *ThreadsActivity;*
- 8 файлів (методів обробки):
    - *ThreadBetterWordCount;*
    - *ThreadSequentialWordCount;*
    - *RxTwoThreadsWordCount;*
    - *RxParallelStreamWordCount;*
    - *RxSequentialWordCount;*
    - *CoroutinesTwoThreadsWordCount;*
    - *CoroutinesSequentialWordCount;*
    - *CoroutinesTwoThreadsWordCount;*
- Моделі та утіліти;
- Ресурси та екрани.

3.14. Програмні інтерфейси

Точки входу в додаток в MainActivity, який, безумовно, визначено в Manifest.xml. Клас Activity має дуже специфічний життєвий цикл у системі Android (див. Додаток Е). MainActivity тримає шар *activity_layout*, у якому присутня клавіша з ідентифікатором «*bthThreads*», для якого були визначені дві дії:

- Першим чином перевіряємо дозвіл системи на користування зовнішнім носієм;
- Далі показ *ThreadsActivity*

ThreadsActivity включає в себе розгортання під назвою *activity_threads*, які, в свою чергу, включають текстові поля з описом і 8 кнопок, кожна з яких викликає в діалоговому вікні завантаження і початок процесу обробки.

Перші два методи – це використання Threads (в один потік та два потоки). Назви файлів:

- Файл *ThreadSequentialWordCount;*
- Файл *ThreadTwoThreadsWordCount*.

Три методи – це використання RxJava (в один потік, у два потоки, паралельна обробка). Файли:

- Файл *RxSequentialWordCount;*
- Файл *RxTwoThreadsWordCount;*
- Файл *RxParallelStreamWordCount.*

Три методи – це використання Kotlin Coroutines (у 4 потоки, один потік, у два потоки). Файли:

- Файл *CoroutinesBetterWordCount;*
- Файл *CoroutinesSequentialWordCount;*
- Файл *CoroutinesTwoThreadsWordCount.*

Розроблено утіліту *LoadingViewMaterial* для перекриття UI-потоку до того, як закінчиться.

3.15. Інструментарій розробки, мова програмування

- Мови: Java, Kotlin, XML
    - Implementation Threads;
    - Implementation RxJava;
    - Implementation Kotlin Coroutines;
    - Implementation EventBus;
    - Implementation Support-Compat;
- Інструментарій: Android Studio

Висновок до розділу

У цьому розділі розглядається створення додатків, реалізація інтерфейсів взаємодії з користувачем, описані інструменти, бібліотеки та мови програмування, а також інструментарій для розробки методів отримання необхідних дозволів із системи, класів і функцій для читання XML-файлу на зовнішньому носії, а також робота над його контентом.

Крім того, розглядаються принципи дизайну інтерфейсу користувача.

Визначте склад бібліотеки, яка була протестована додатком.

4. ОЦІНКА СТВОРЕНИХ МЕТОДІВ ПАРСИНГУ ТА ЇХ ПОРІВНЯННЯ

4.1. Аналізу навантаженності на систему від реалізованих методів

З IDE Android Studio розробники додатків на МОС Андроїд мають можливість побачити, які ресурси споживає їх програмний продукт. Така функція називається Profiler. Це дає нам можливість достучатися до таких параметрів гаджета, як споживання енергії, оперативна пам'ять, процесор та мережа.

Для кожного параметра малюються графіки, які змінюються з часом.

Для процесора можна спостерігати у відсотках від його навантаження. Для пам'яті – його споживання в мегабайтах. Для енергоспоживання параметрами є низький, середній і важкий. У цьому дослідженні параметр мережі нас не цікавить, тому що немає інтернет-запитів, які ми робимо. Приклад такого аналізу використання ресурсів програми можна побачити на рис. 4.1.

Алгоритм такої оцінки простий, запустіть програму, підключіть телефон до комп'ютера, включіть Profiler в Андроїд Студії і знайдіть процес, який нам потрібен (наш додаток). Потім ми спостерігаємо деякий фіксований час, щоб система могла отримати оцінку за усіма спостереженнями.

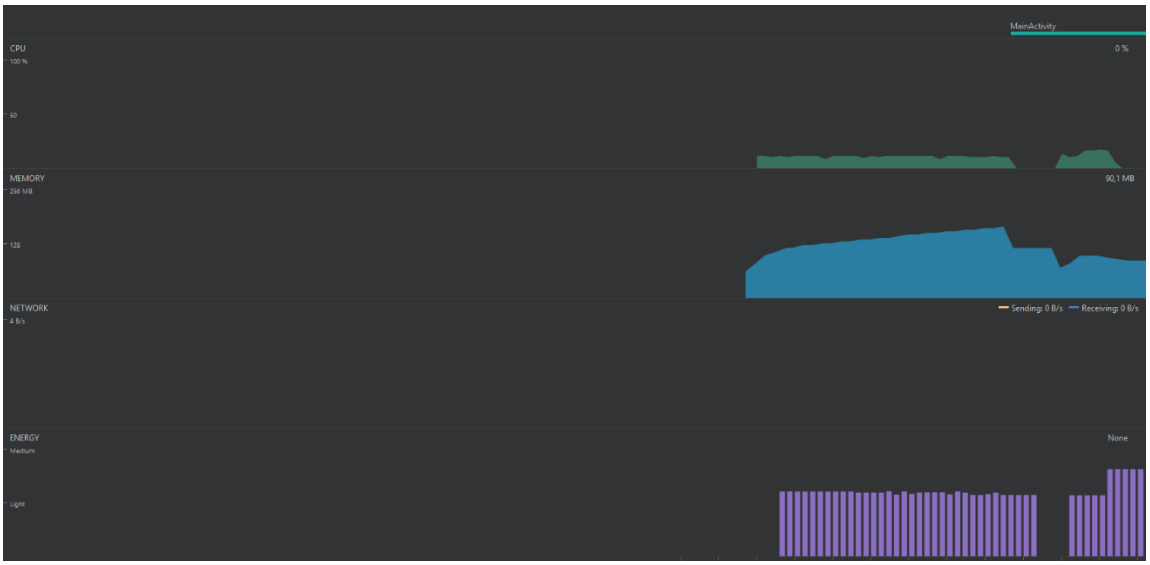


Рисунок 4.1 – Використання ресурсів додатком під час його запуску

4.1.1. Аналіз навантаженності на систему від реалізованих методів за допомогою Threads

При обробці метода Threads and Locks в один потік споживаються певні ресурси (рис. 4.2):

- CPU – 12%;
- Memory – 180.7 мб;
- Energy consumption – medium.

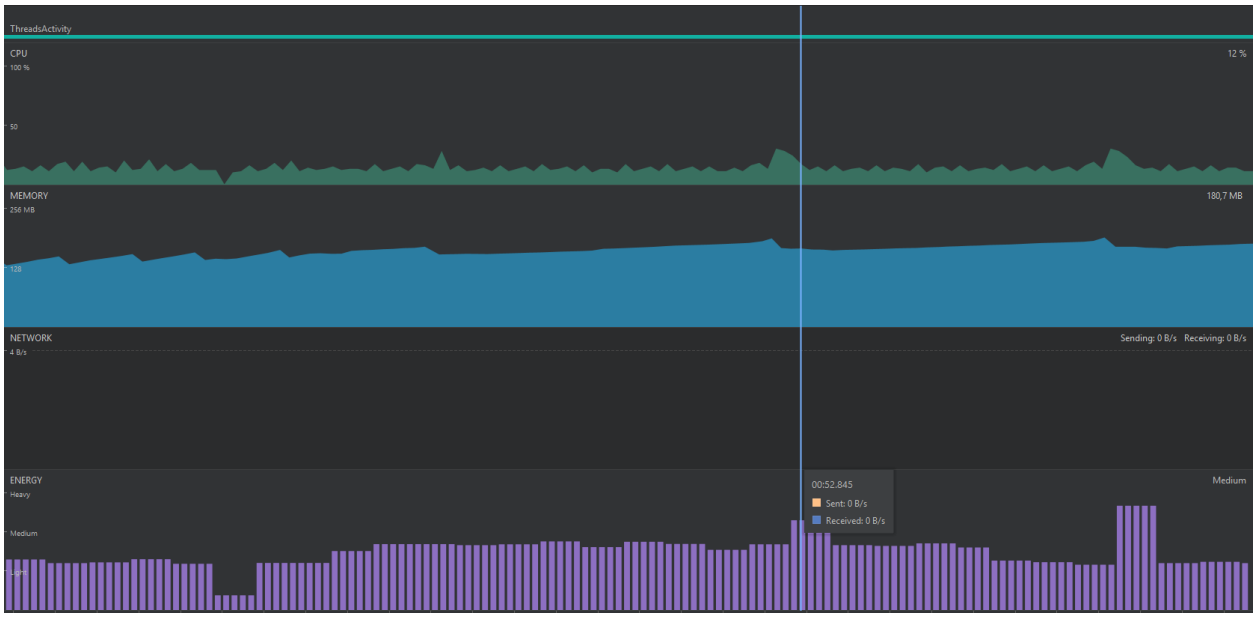


Рисунок 4.2 – Використання методу Threads в один потік

З цього можна зробити висновок, що цей метод не створює великого навантаження на систему і, отже, дійсний для використання в будь-яких обставинах.

При обробці метода Threads and Locks в два потоки споживаються певні ресурси (рис. .4.3):

- CPU – 23%;
- Memory – 233.1 мб;
- Energy consumption – heavy.

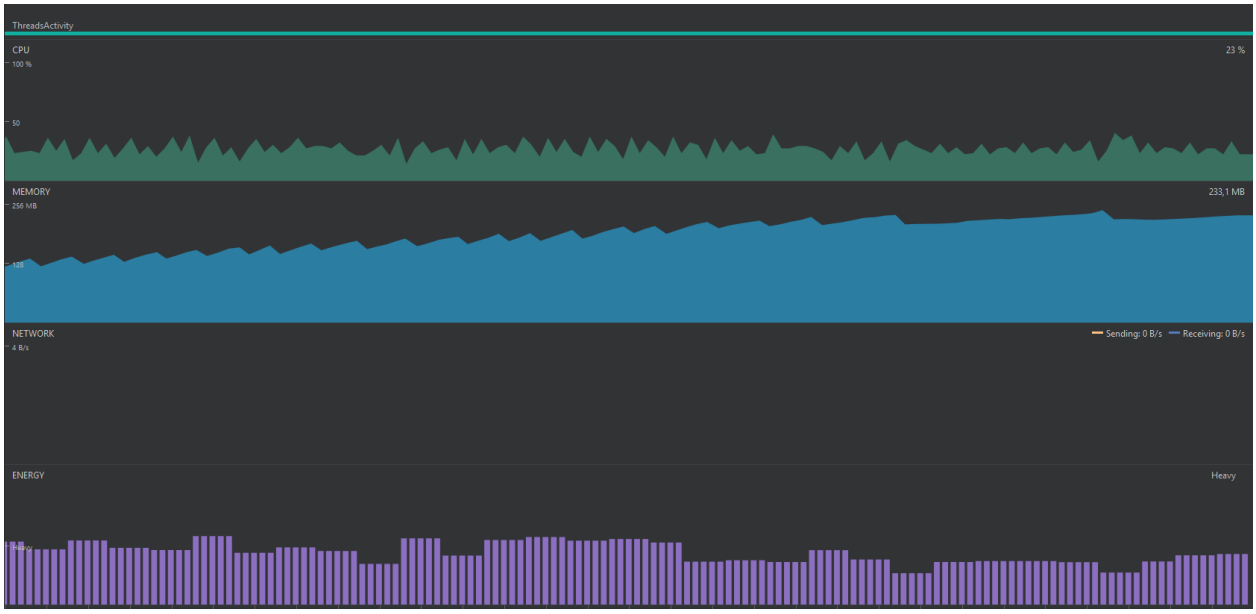


Рисунок 4.3 – Використання методу Threads в два потоки

Цей метод семантичного аналізу створює велике навантаження на систему, і тому користувачі повинні заздалегідь знати, за яких обставин він буде використовуватися для визначення можливості його інтеграції.

4.1.2. Аналіз навантаженності на систему від реалізованих методів за допомогою RxJava

При обробці метода RxJava в один потік споживаються певні ресурси (рис. 4.4):

- CPU – 15%;
- Memory – 204.6 мб;
- Energy consumption –medium.

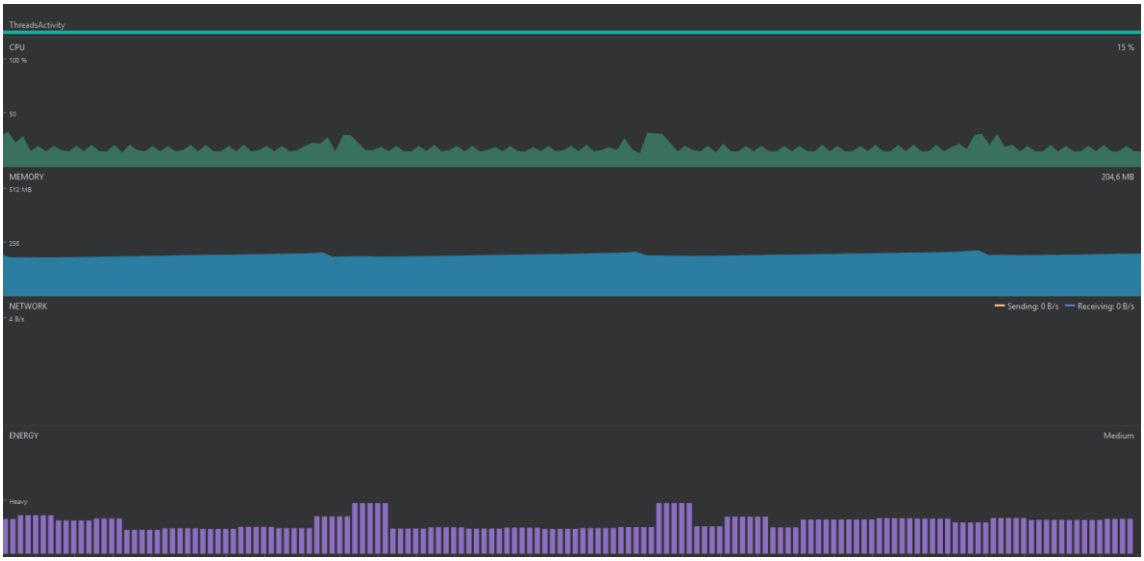


Рисунок 4.4 – Використання методу RxJava в один потік

Цей метод обробки файлу ММЛ не займає багато часу для системи і, отже, є дійсним для використання в будь-яких обставинах.

При обробці метода RxJava в два потоки споживаються певні ресурси (рис. 4.5):

- CPU – 39%;
- Memory – 228.6 мб;
- Energy consumption – heavy.

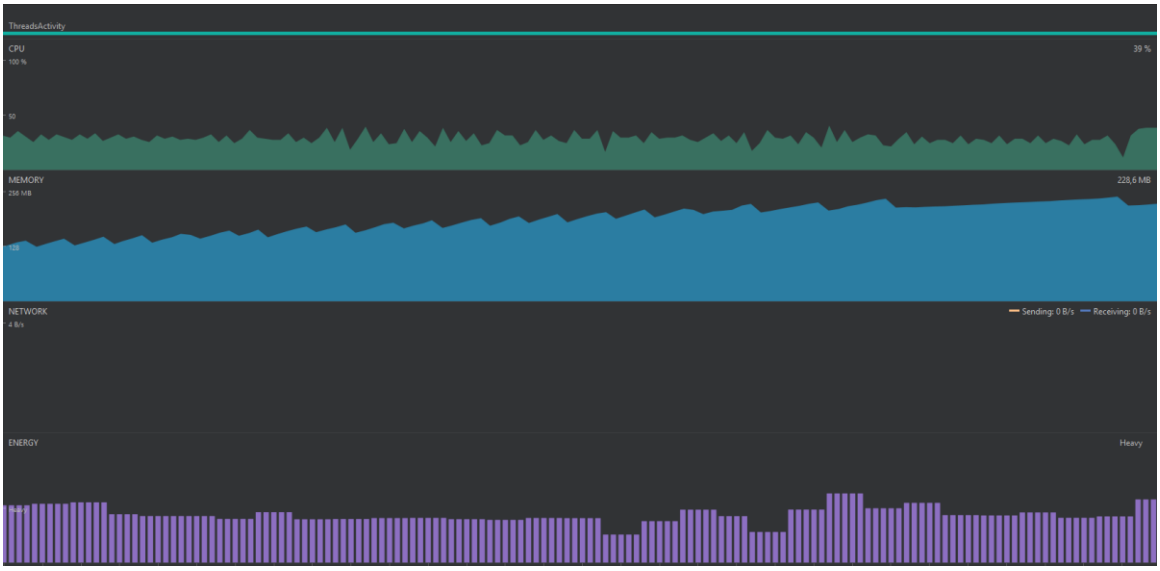


Рисунок 4.5 – Використання методу RxJava в один потік

Данний метод обробки споживає значну кількість системних ресурсів і, отже, дійсний за раніше видатних обставин програми.

При обробці метода RxJava за паралельними потоками споживаються певні ресурси. (рис. 4.6):

- CPU – 20%;
- Memory – 179.7 мб;
- Energy consumption – medium.

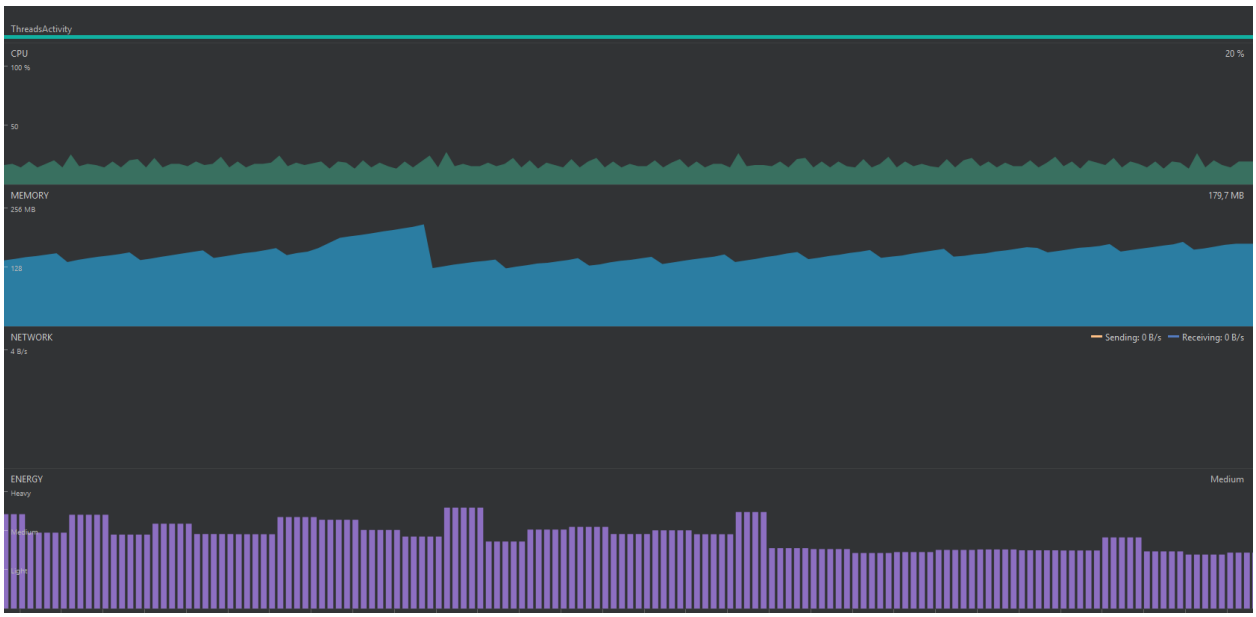


Рисунок 4.6 – Використання методу RxJava з паралельними потоками

Цей метод обробки злегка завантажує систему і тому рекомендується як для потужних віджетів, так і для слабких.

4.1.3. Аналіз навантаженності на систему від реалізованих методів за допомогою Kotlin Coroutines

При обробці метода Kotlin Coroutines в один потік споживаються певні ресурси (рис. 4.7):

- CPU – 21%;
- CPU – 163.7 мб;
- Energy consumption – heavy.

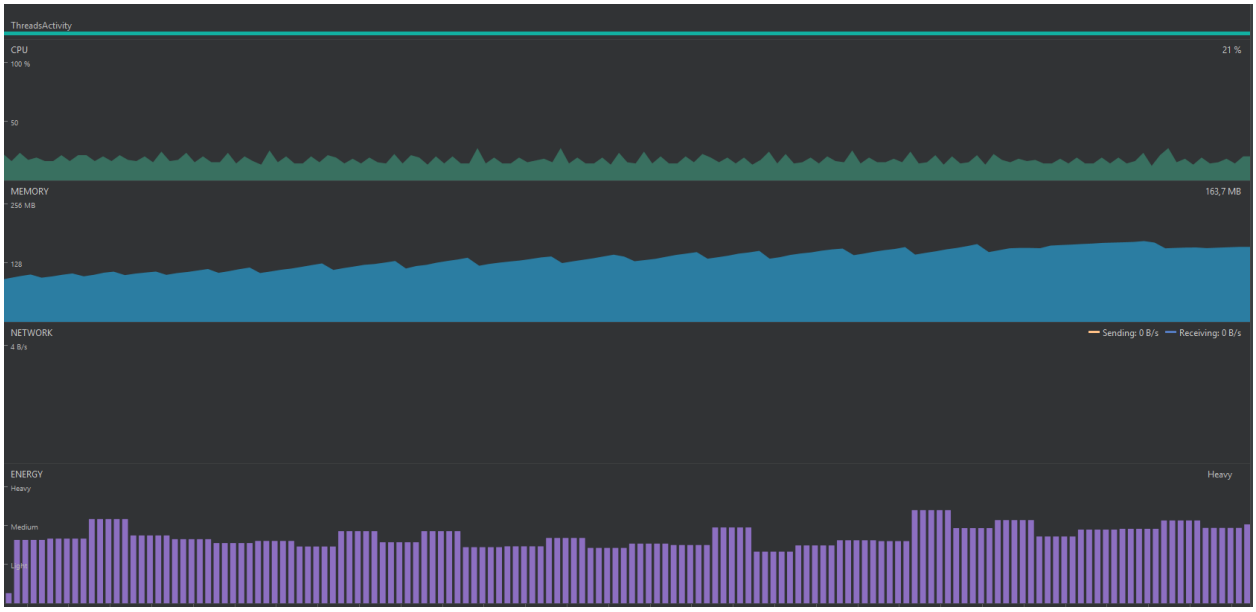


Рисунок 4.7 – Використання методу Kotlin Coroutines в один потік

Цей метод обробки завантажує систему менше для попередніх, але слід зазначити, що споживання енергії знаходиться на рівні «важко».

При обробці метода Kotlin Coroutines в два потоки споживаються певні ресурси (рис. 4.8):

- CPU – 31%;
- Memory – 238.4 мб;
- Energy consumption – heavy.

Цей метод обробки менше завантажує систему більше, ніж попередню, а споживання енергії залишається на тому ж рівні, тому цей метод повинен використовуватися на більш потужних девайсах.

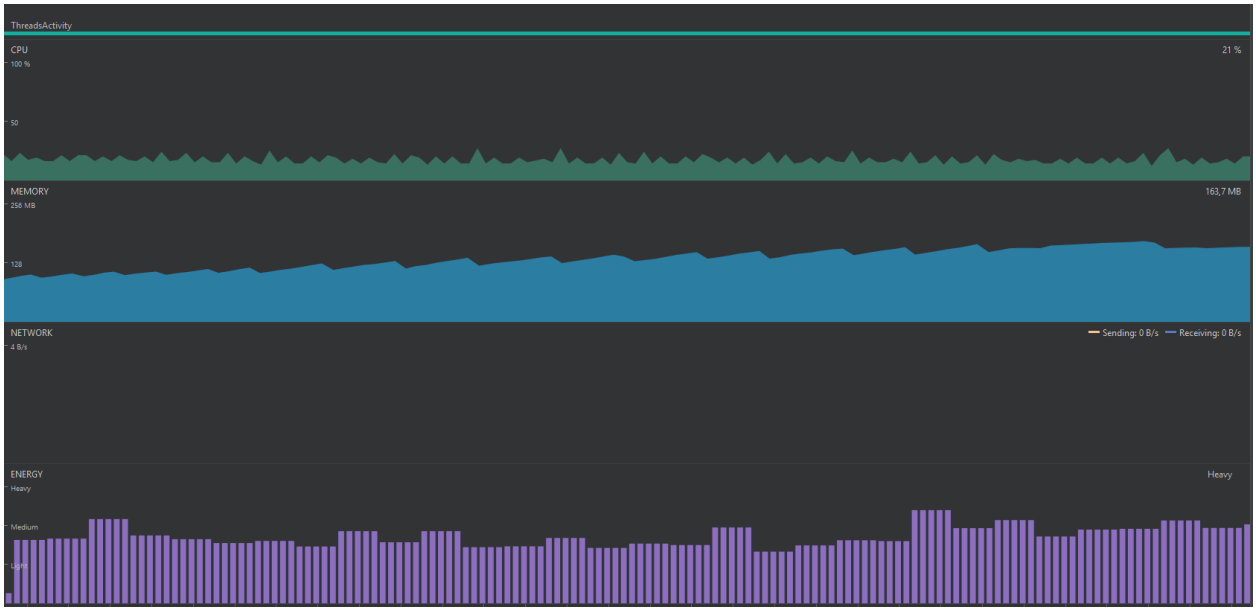


Рисунок 4.8 – Використання методу Kotlin Coroutines в два потоки

При обробці метода Kotlin Coroutines в чотири потоки споживаються певні ресурси (рис. 4.9):

- CPU – 63%;
- Memory – 316.4 мб;
- Energy consumption – heavy.

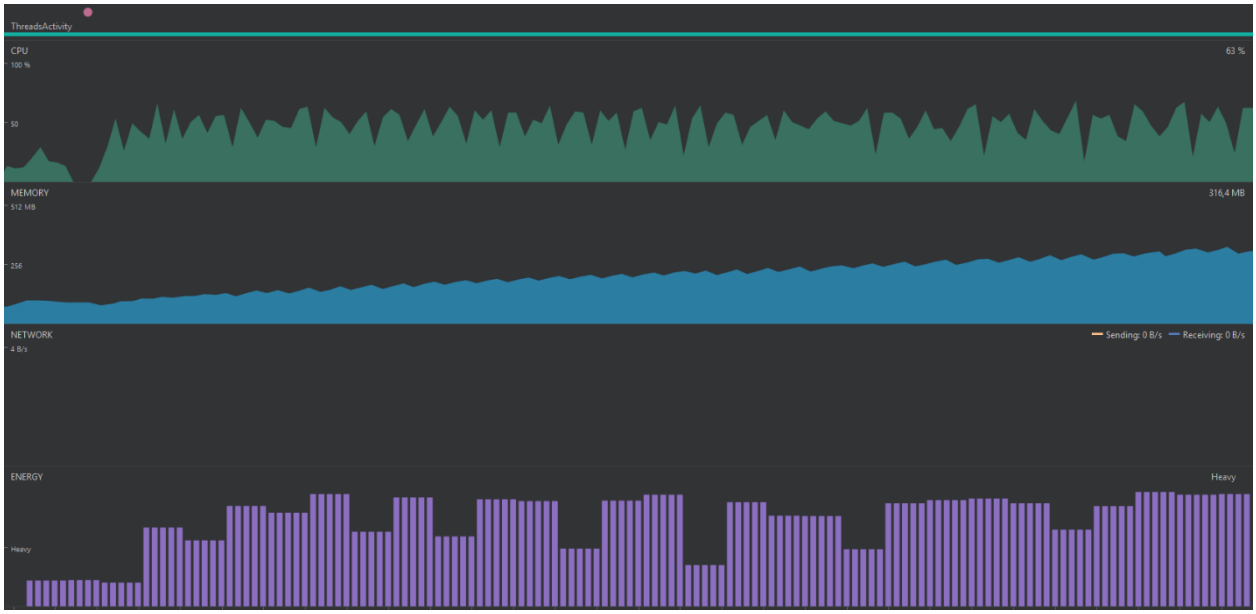


Рисунок 4.9 – Використання методу Kotlin Coroutines в чотири потоки

Цей метод споживає занадто багато ресурсів і енергії, тому його краще використовувати на потужних гаджетах і в ситуаціях, коли ви не можете обійтися без нього.

4.2. Аналізу витраченого часу на завершення парсингу створеними методами

Для повного аналізу часових витрат на обробку реалізованих методів семантичного аналізу виводиться наступний алгоритм:

1) фіксована кількість разів визначає час для повної обробки файлів XML без завантаження системи і створення зразка. Під терміном «без навантаження» мова йде про тестування методу відразу після перезавантаження системи, коли не всі служби резервного копіювання почали працювати;
2) фіксована кількість разів визначає час для повної обробки файлів XML із завантаженням системи і створення зразка. Під терміном «з навантаженням» мова йде про тестування методу після включення декількох ресурсоємних додатків;
3) візьміть середнє значення з двох зразків. Ці два значення будуть нашими можливими часовими інтервалами, які необхідно витратити на обробку певним методом.

4.2.1. Аналіз потрібного часу на завершення методів, реалізованих за допомогою Threads

Проведемо аналіз методу обробки Threads і зафіксуємо середні значення (рис. 4.10):

- Один потік – 30,67 сек;
- Два потоки – 24,50 сек;
- Один потік з додатковим навантаженням – 33,37 сек;
- Два потоки з додатковим навантаженням – 25,56 сек.

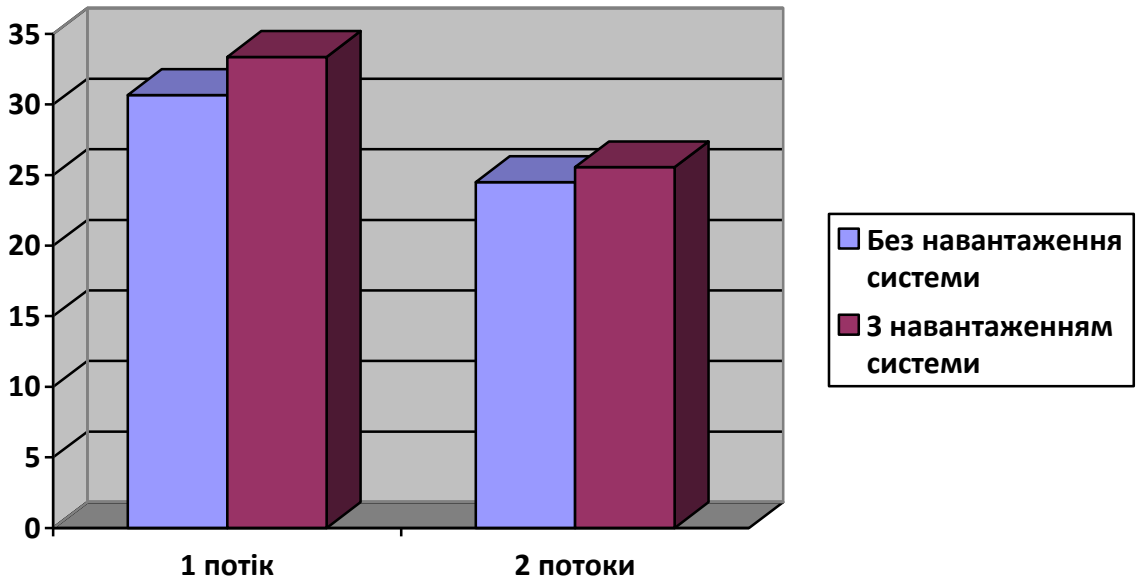


Рисунок 4.10 – Діаграма часу при обробці за допомогою Threads

4.2.2. Аналіз потрібного часу на завершення методів, реалізованих за допомогою RxJava

Проведемо аналіз методу обробки RxJava і зафіксуємо середні значення (рис. 4.11):

- Один потік – 30,26 сек;
- Два потоки – 23,23 сек;
- Паралельні потоки – 13,05 сек;
- Один потік з додатковим навантаженням – 32,14 сек;
- Два потоки з додатковим навантаженням – 24,53 сек;
- Паралельні потоки з додатковим навантаженням – 14,44 сек.

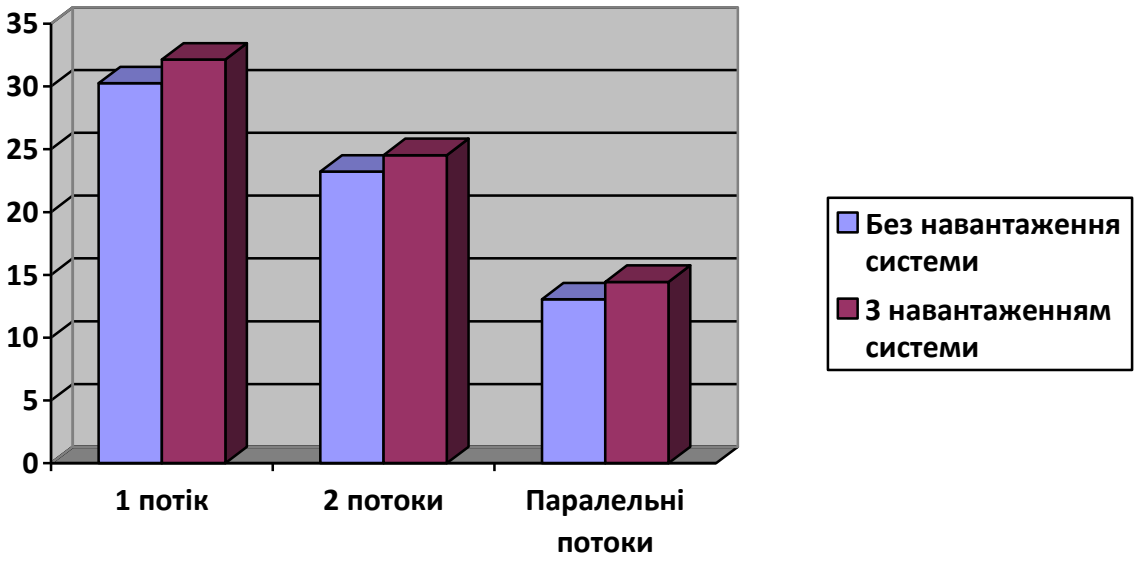


Рисунок 4.11 – Діаграма часу при обробці за допомогою RxJava

4.2.3. Аналіз потрібного часу на завершення методів, реалізованих за допомогою Kotlin Coroutines

Проведемо аналіз методу обробки Kotlin Coroutines і зафіксуємо середні значення (рис. 4.12):

- Один потік – 30,09 сек;
- Два потоки – 22,57 сек;
- Чотири потоки – 16,47 сек; .
- Один потік з додатковим навантаженням – 31,99 сек;
- Два потоки з додатковим навантаженням – 23,45 сек;
- Чотири потоки з додатковим навантаженням – 18,57 сек.

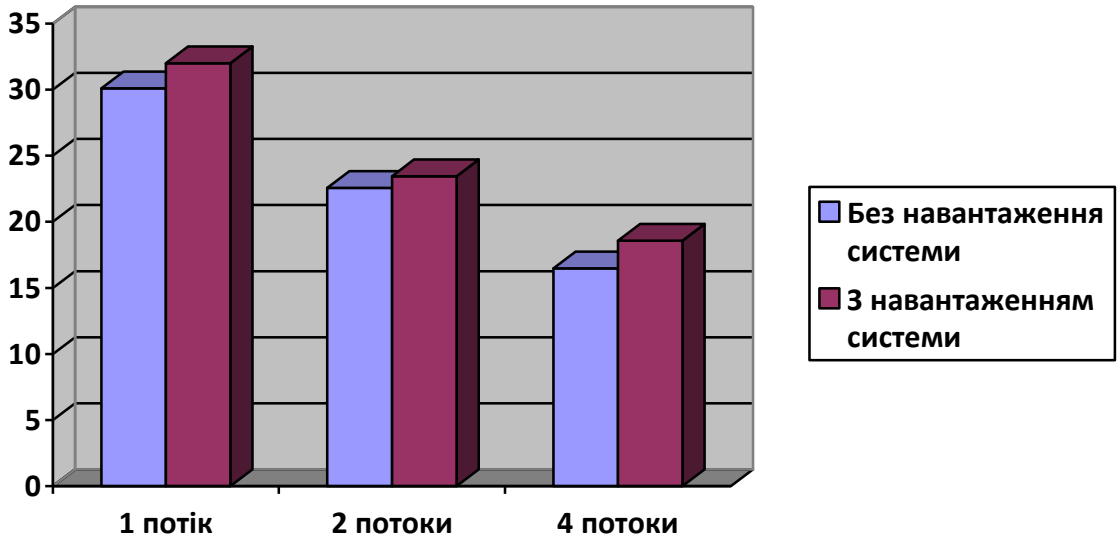


Рисунок 4.12 – Діаграма часу при обробці за допомогою Kotlin Coroutines

4.2.4. Загальне порівняння усіх методів за часом

З усіма фіксованими тестовими значеннями з завантаженою системою і без навантаження можна провести повне порівняння (рис. 4.13), всіх реалізованих методів, а також трьох раніше обраних бібліотек під час аналізу задачі: XMLPullParser, SAXParser, DOMParser.

Дивлячись на тимчасові витрати застосовуваних методів і системних ресурсів, які споживаються, можна побачити, що чим більше потоків розподілено, тим більше навантаження на систему. Але зовсім інша ситуація у метода RxJava з паралельними потоками. В даному випадку використовується функція *parallelStream*(). З її допомогою сама система створює необхідну кількість потоків, що залежать від багатоядерності процесора гаджета. Тому даний метод не дає жодних переваг при використанні на слабких пристроях.

Для всіх обставин, при яких необхідно буде виконати XML парсинг в додатку, ви можете вибрати найбільш підходящий метод і використовувати його.

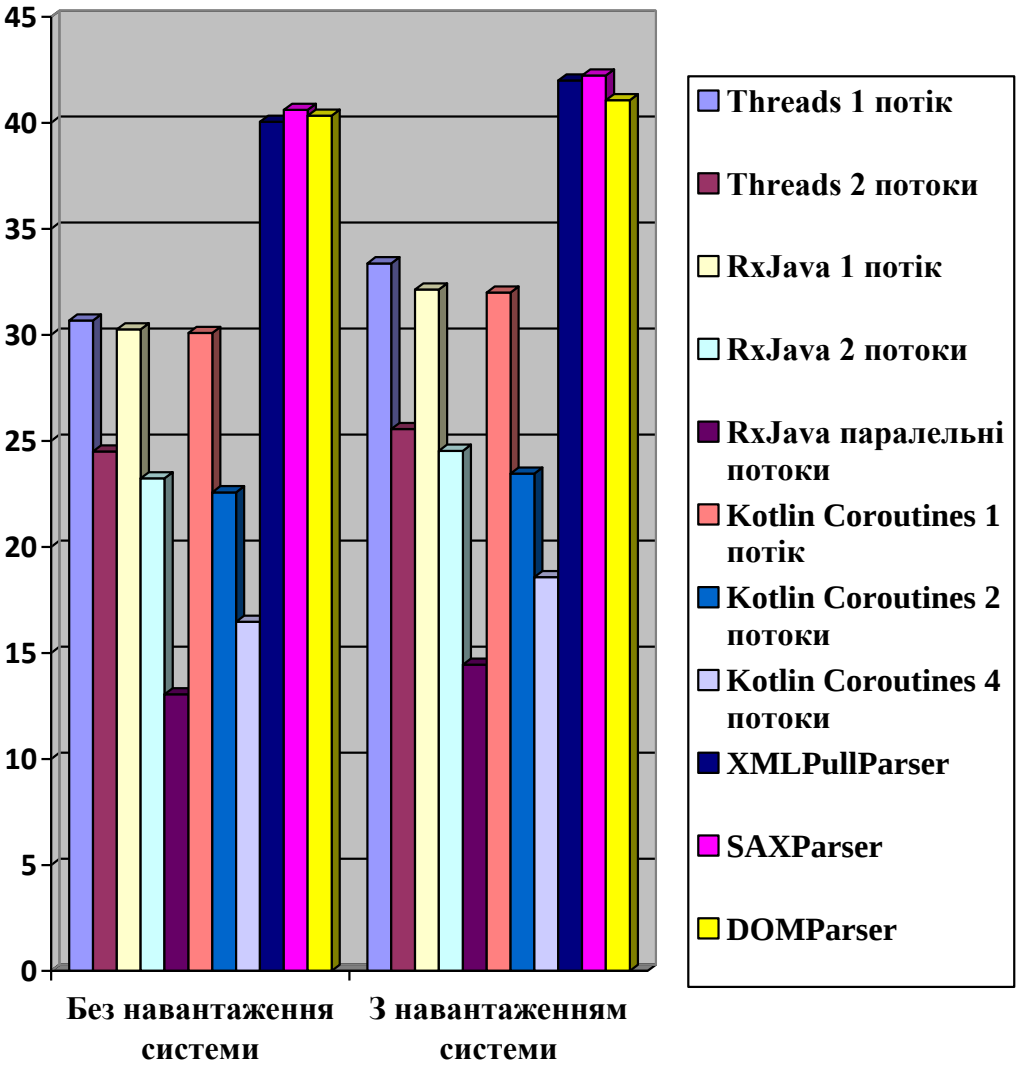


Рисунок 4.13 – Діаграма порівняння всіх реалізованих методів та трьох обраних існуючих бібліотек

4.3. Створення алгоритму для визначення найоптимальнішого методу обробки даних.

Дослідивши розроблені методи семантичного аналізу, зібравши усі дані та порівнявши їх між собою та з уже існуючими бібліотеками можна тепер створити алгоритм для пошуку найоптимальнішої поведінки у додатку.

Для початку, слід розібратися, які рішення та умови будуть прийматися для побудови алгоритму. Також будемо реалізовувати усі етапи алгоритму у вигляді блок-схеми.

Першим кроком буде питання «Чи може алгоритм обробки навантажувати пристрій?». Представимо його у вигляді умови (рис. 4.14).

Рисунок 4.14 – Вхідна умова «Чи може алгоритм обробки навантажувати пристрій?»

При виборі відповіді «Ні» рухаємося до наступної умови «Чи наявний в девайсі багатоядерний процесор?» (див. рис 4.15). При відповіді «Так» ми можемо спокійно запускати метод «RxParallelStreamWordCount», оскільки він ідеально вписується в дану модель: швидкий, не несе навантаження на систему та добро поводить себе на багатоядерних пристроях.

Рисунок 4.15 – Умова «Чи наявний в девайсі багатоядерний процесор?»

Якщо на умову «Чи може алгоритм обробки навантажувати пристрій?» ми отримаємо відповідь «Ні», то рухаємося до наступної умови «Чи цікавить нас найшвидший метод?» (див. рис. 4.16).

Рисунок 4.16 – Умова «Чи цікавить нас найшвидший метод?»

При виборі відповіді «Так» маємо метод «RxSequentialWordCount», оскільки він ідеально підходить для одноядерного процесора та швидко парсить дані.

При виборі відповіді «Ні» йдемо до наступної умови «Чи великий за об’ємом файл?» (див. рис. 4.17). Якщо файл більший за 200мб – то єдиний метод який нам підходить – це «ThreadsSequentialWordCount». Якщо файл менший – то в таком випадку можна обрати будь-який з методів вже існуючих бібліотек: «XMLPullParser», «DOMParser», «SAXParser».

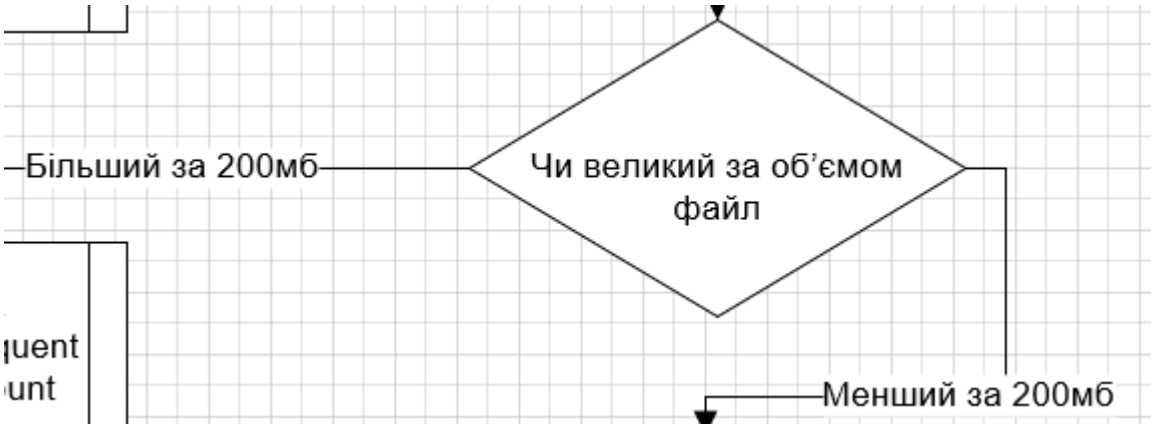


Рисунок 4.17 – Умова «Чи великий за об’ємом файл?»

Повернемося до вхідної умови «Чи може алгоритм обробки навантажувати пристрій?». При виборі відповіді «Так» нас очікує більш різноманітніший цикл.

Наступним питанням буде «Чи наявний в девайсі багатоядерний процесор» (ця умова була вже раніше – див. рис. 4.15).

При виборі відповіді «Ні» - запускаємо метод «CoroutinesSequentialWordCount», який ідеально підходить у випадку, коли ми можем навантажити систему, але гаджет може похизуватися лише одноядерним процесором.

При виборі відповіді «Так» йдемо до наступної умови «Скільки пристрій може виділити пам’яті на один процес?» (див. рис. 4.18).

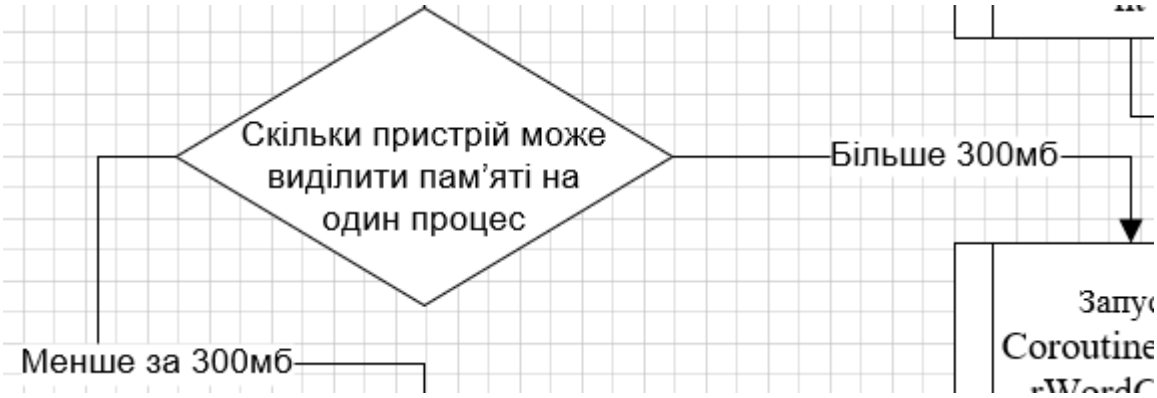


Рисунок 4.18 – Умова «Скільки пристрій може виділити пам’яті на один процес?»

При виборі відповіді «Більше 300мб» можна сміливо запускати метод «CoroutinesBetterWordCount», який споживає дуже багато ресурсів, але ідеально виконує свою роботу.

За іншого розкладу – рухаємося до наступної умови «Кількість ядер» (див. рис. 4.19). Якщо у нас більше 4 ядер – то знову можна запустити метод «RxParallelStreamWordCount», який має більше потужність при більшій кількості ядер.

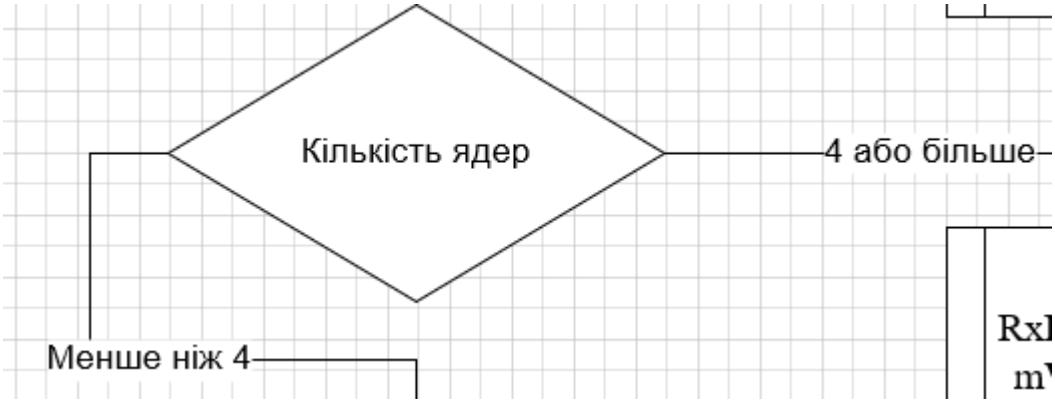


Рисунок 4.19 – Умова «Кількість ядер»

В протилежному разі рухаємося до умови «Чи навантажена система у даний момент?» (див. рис. 4.20). При відповіді «Так» - викликаємо метод «CoroutinesTwoThreadsWordCount», який краще всіх виконує роботу при доповненому навантаження системи іншими процесами.

Рисунок 4.20 – Умова «Чи навантажена система у даний момент?»

При відповіді «Ні» рухаємося до останньої умови «Заряд батареї» (див. рис. 4.21). Якщо заряд менший за 30% – слід викликати метод «ThreadTwoThreadsWordCount», інакше – «RxTwoThreadsWordCount», який споживає занадто багато енергії девайса, але виконує свою роботу дуже добре.

Таким чином, ми розробили алгоритм, який проходячи декілька умов може визначити найоптимальніший метод семантичного аналізу для конкретного девайса при конкретних умовах (див. Додаток Є).

Рисунок 4.21 – Умова «Заряд батареї»

Висновок до розділу

У цьому розділі аналізується навантаження на систему, яку створює кожен з методів, а також витрати часу на повну обробку тих же методів. Певна

залежність від використання n-ої кількості потоків для споживання системних ресурсів. Після всіх тестів було розроблено унікальний алгоритм для визначення найоптимальнішого методу серед розглянутих та створених.

5. МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП-ПРОЄКТУ

5.1. Маркетинговий опис ідеї проекту

Маркетинговий опис ідеї проекту зазначену в табл. 5.1.

Таблиця 5.1. Основний опис ідеї стартап-проекту

Зміст основної ідеї	Основні напрямки застосування	Основні вигоди для користувача
Цілі: Розробити оптимізовані методі обробки великих об’ємів даних (XML-файлів) для різних ситуацій та представити їх у SDK. Таким чином інші розробники, для яких придбають даний продукт, зможуть з легкістю маніпулювати вже створеними методами та швидко обробляти дані у своїх мобільних додатках. **Завдання:** Розробити алгоритми парсингу, імплентувати їх у SDK, протестувати, при негативному результаті – покращити їх. Зробити документацію по використанню SDK та зарефакторити код таким чином, щоб він був зрозумілим для усіх.	1. Додатки з використання великого обсягу даних. 2. Додатки-клієнти для систем розумних будинків. 3. Додатки з великою кількістю клієнтів, за допомогою яких розробники хочуть зняти навантаження з серверів.	1. Швидкість роботи. 2. Оптимізоване навантаження. 3. Оптимізована витрата ресурсів мобільного девайса.

Далі слід визначити слабкі, нейтральні та сильні сторони проекту у порівнянні з обраними конкурентами (див. табл. 5.2).

Таблиця 5.2. Сильні, нейтральні та слабкі характеристики проекту

№	Техніко-економічні характеристики ідеї	Продукція конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	XML PullParser	DOM Parser	SAX Parser			
1	Максимальний об'єм вхідних данних – без лімітів	+						+
2	Швидка обробка	+	+					+
3	Швидка обробка при паралельному навантаженні	+					+	
4	Оптимізація навантаження на систему	+		+	+		+	
5	Мінімальна версія Андроїд – API 14	+	+	+		+		

5.2. Технологічний аудит маркетингової ідеї проекту

Слід провести аудит технологій, за допомогою яких будемо реалізовувати проект (див. табл. 5.3).

Таблиця 5.3. Технологічна здійсненність проєкту

№	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
1.	Бібліотека для роботи з багатопоточністю	RxJava	+	+ (безкоштовно)
		Kotlin Coroutines	+	+ (безкоштовно)
		Threads	+	+ (безкоштовно)
2.	Мова програмування	Kotlin	+	+ (безкоштовно)
		Java	+	+ (безкоштовно)
Для реалізації проекту були обрані усі техногології для багатопоточного програмування та мова програмування Kotlin.				

5.3. Аналіз маркетингових можливостей старту стартап-проекту

Таблиця 5.4. Попередня характеристика потенційного маркетингового ринку стартапу

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних конкурентів, од	1 (нативні методи ідуть в комплектації з мовою програмування)
2	Загальний об’єм продаж, грн/ум.од	Нативні методи є безкоштовними та ідуть в комплектації з мовою програмування
3	Динаміка маркетингового ринку (якісна оцінка)	Стагнує
4	Наявність лімітів для входу (вказати характер обмежень)	Можливість інтеграції з продуктами покупців

Закінчення таблиці 5.4

5	Рідкі вимоги до сертифікації	Немає, оскільки цей ринок не контролюється якимись органами держави
6	Середня норма рентабельності (або по ринку), %	20% на рік

Таблиця 5.5. Характеристика потенційних маркетингових клієнтів стартап-проекту

№ п/п	Потреба, що створює ринок	Цільова аудиторія (сегменти ринку)	Різниця у поведінці різних можливих цільових груп клієнтів	Вимоги користувачів до товару
	Потреба швидко, якісно оброблювати інформацію та дані, при цьому не навантажуючи систему	IT-компанії з розробки мобільних додатків для інтеграції з розумними домами	Клієнти прагнуть покращувати свій продукт для конкуренції на ринку, тому вони зацікавленні у постійній підтримці (саппорт) та можливості інновацій.	1. наявність Android-додатки; 2. робота з великими об’ємами данних.

Таблиця 5.6. Фактори загроз

№ п/п	Фактор загрози	Зміст загрози	Реакція компанії
	Вихід конкурентів на ринок	Після виходу продукту на ринок можлива поява конкурентів	Нові інновації для приваблення нових клієнтів

Закінчення таблиці 5.6

	Неготовність розробників використовувати зовнішні бібліотеки	Не усі розробники будуть прагнути використовувати чужі SDK через можливу складність інтеграції	Створення дуже чіткої документації по усім крокам інтеграції продукту
	Можливість розробників вирішити проблеми з парсингом власноруч	Деякі компанії можуть вирішити, що вони здатні побороти проблему парсингу великого об’єму даних власноруч без сторонніх бібліотек	Створення дуже великого функціоналу та представлення його таким чином, щоб компанії могли зрозуміти, що купуючи наш продукт – вони економлять багато часу та грошей на власні розробки

Таблиця 5.7. Фактори можливостей

№ п/п	Фактор можливості	Зміст можливості	Реакція компанії
	Нові сфери використання	Крім інтеграції з розумними домами можуть з’явитися й нові можливості	Націлення на нові сфери використання для приваблення більшої кількості клієнтів
	Інтеграція на нові платформи	Можливість використовувати бібліотеку не тільки на платформі Андроид, а й на ИОС	Приваблення нових розробників для інтеграції з новою платформою, розростання компанії
	Інтеграція с серверами	Вивід парсингу даних на ремоут (сервер) для поліпшення інтеграції с будь-якою існуючою платформою	Перехід з розробки під одну платформу до розробки на сервері

Припустимо, що конкурентом будуть нативні методи обробки даних.

Таблиця 5.8. Ступеневий аналіз можливої конкуренції на ринку

Характеристики конкурентного середовища	В чому визначається дана характеристика	Можливий плив на діяльність підприємства
1. Вказати тип конкуренції - чиста	Один продавець пропонує товар, який не має близьких замінників.	Пропонує нові технології та методи обробки, що є швидшими та оптимізованішими
2. За рівнем конкурентної боротьби - міжнаціональний	Ринок орієнтований на компанії по усьому світу	Підтримка на найпопулярніших мовах світу (в майбутньому – на усіх мовах клієнтів)
3. За галузевою ознакою - внутрішньогалузева	Розробку можна використовувати тільки при розробці андроїд додатків та в інтеграції з розумними домами	Можливий вихід у майбутньому на нові сфери використання обробки інформації
4. Конкуренція за видами товарів: товарно-видова	Усі методи виконують одну функції – оброблюють (парсять) файли	Можливість покращити та оптимізувати функціонал якомога найкраще
5. За характером конкурентних переваг нецінова	Нативні методи не є платними (ідуть у коробці з мовами програмування)	Нативні методи не є оптимізованими – тому перехід клієнтів на продукт є лише питанням часу
6. За інтенсивністю - не марочна	Існуючі нативні методи нікому не належать	Розвиток та гарна рекламна компанія для світової впізнаваності

Таблиця 5.9. Аналіз конкуренції за М. Портером

Складові аналізу	Прямі опоненти в галузі	Потенційні опоненти	Постачальники	Клієнти	Замінники
	Нативні методи обробки	Бар'єром входження є неготовність клієнтів використовувати стороннє SDK	Постачальники мають вміти доставляти бібліотеку за допомогою імпортів	Споживачі повинні весь час користуватися функціями саппорту для якісної інтеграції	Замінники є безкоштовними
Висновки:	Інтенсивність не є великою, оскільки функціональні можливості конкурента значно гірші	Моєливості входу є, оскільки за допомогою служби саппорт можна значно поліпшити процес інтеграції. Строки виходу на ринок: одразу після релізу готового продукту	Не диктують	Не диктують	Серед обмежень можна відмітити небезкоштовність товару

Таблиця 5.10. Обґрунтування різких факторів конкурентоспроможності

№ п/п	Фактор можливої конкурентоспроможності	Обґрунтування
	Функціональність	У конкурента дуже дрібна функціональність, що не може конкурувати з функціональністю даного продукту
	Швидкість обробки	У конкурента не оптимізовані методи та алгоритми обробки, що робить парсинг дуже повільним
	Навантаженість на систему	У конкурента не оптимізовані методи та алгоритми обробки, що робить парсинг дуже затратним для системи
	Максимальний об’єм для обробки	Методи обробки конкурента не можуть обробляти великі масиви інформації

Таблиця 5.11. Порівняльний аналіз сильних та слабких сторін «MultiParser»

№ п/п	Фактор можливої конкурентоспроможності	Бали 1-20	Рейтинг конкурентів у порівнянні з «MultiParser»						
			Мінус 3	Мінус 2	Мінус 1	0	Плюс 1	Плюс 2	Плюс 3
1	Функціональність	18	*						
2	Швидкість обробки	20	*						
3	Навантаженість на систему	12			*				
4	Максимальний об’єм для обробки	10		*					

Таблиця 5.12. SWOT- аналіз проекту

Сильні сторони: Висока функціональність Висока швидкість парсингу Оптимізованість методів	**Слабкі сторони:** Вихід конкурентів на ринок Неготовність розробників використовувати зовнішні бібліотеки
Можливості: Нові сфери використання	**Загрози:** Складний процес інтеграції з продуктом

Таблиця 5.13. Альтернативи впровадження проекту

№ п/п	Альтернатива ринкової поведінки	Ймовірність можливого отримання ресурсів	Строки реалізації
	Створення максимально оптимізованих методів обробки	Висока ймовірність (80%)	Великі строки реалізації (2 місяці)
	Потужна реклама продукту, яка зацікавлює клієнтів	Мала ймовірність (30%)	Середні строки реалізації (1 місяць)
	Пропонування нових сфер використання продукту	Висока ймовірність (90%)	Середня строки реалізації (1 місяць)

Було обрано альтернативу: пропонування нових сфер використання продукту (не тільки інтеграція з розумними домами). Оскільки за строками реалізації і за ймовірністю отримання ресурсів вона є найоптимізованішою.

5.4. Ринкова стратегія ринку

Таблиця 5.14. Вибір цільових груп можливих споживачів

№ п/п	Опис цільової групи можливих клієнтів	Готовність користувачів сприйняти продукт	Попит в межах орієнтовної групи	Інтенсивність конкуренції	Простота входу
	Компанії, що займаються розробкою мобільних додатків для інтеграції з розумними будинками	Продукт є поліпшенням існуючих варіантів – а тому готовність прийняття має бути високою	Через малу функціональність альтернатив попит на цей продукт має бути високим	Конкуренції не існує, інують тільки нативні методи, інтенсивність дуже мала	Готовність інших компаній використовувати моє SDK – цей фактор може завадити просто входу в сегмент
Які цільові групи обрано: ІТ-компанії з розробки мобільних додатків					

Таблиця 5.15. Обрання базової стратегії розвитку проекту

№ п/п	Визначена альтернатива розвитку	Стратегія охоплення ринку проектів	Ключові конкурентоспроможні варінти відповідно до обраної альтернативи	Обрана базова стратегія розвитку
	Випустити першу версію продукту з усім запланованим функціоналом	Пропозиція найбільш оптимізованих методів обробки інформації серед існуючих (ставка на універсальність)	Співвідношення простоти та функціональності	Стратегія диференціації

Таблиця 5.16. Обрання базової стратегії конкурентної поведінки

№ п/п	Проект «першопроходець» на ринку?	Компанія буде шукати нових користувачів, або забирати існуючих у опонентів?	Компанія буде копіювати основні можливості товару опонента, і які?	Визначена стратегія конкурентної поведінки
	Так	На 65% - нові, на 35% - старі	Не буде	Стратегія лідера

Таблиця 5.17. Визначення основної стратегії позиціонування

№ п/п	Основні вимоги до товару аудиторії	Основна стратегія розвитку	Основні конкурентоспроможні позиції власного проекту	Вибір асоціацій, що мають сформувати позицію проекту (три ключових)
	Функціональні можливості, швидкість та якість роботи	Стратегія диференціації	Висока швидкість роботи обробки	Висока швидкість, функціональність, підтримка (support)

5.5. Розробка маркетингової програми проекту

Таблиця 5.18. Обрання ключових переваг концепції можливого товару

№ п/п	Потреба	Вигода	Ключові переваги перед опонентами
	Універсальність, швидкість обробки	Висока швидкість та функціональність	Функціональні можливості, оптимізація методів парсингу

Таблиця 5.19. Опис трьох основних рівнів моделі товару

Основні рівні товару	Сутність та складові товару		
I. Товар за задумом	SDK повинно приймати файли без обмежень за розміром, оброблювати 20мб за 18 секунд без навантаження и за 22 з навантаженням, не створювати помітного навантаження на систему та працювати з версією Андроїд API 14.		
II. Товар у реальному виконанні	Основні характеристики	М/Нм	(Вр/Тх/Тл/Е/Ор)
	Швидкість обробки Об’єм інформації Навантаженність Функціональність	М М М М	Тх Тх Тх Тх
	Якість: гарантована висока швидкість обробки та якість (не має норм та стандартів)		
	Пакування: Maven репозиторій		

Закінчення таблиці 5.19

	Марка: MultiPars – MultiParser 
III. Товар із підкріпленням	Існує система саппорту (підтримки) продукту.
	Можливо підключати свої модулі для унікальної обробки даних.
Як товар буде захищено від копіювання конкурентами: за рахунок універсальності та новітніх ідей для створення алгоритмів парсингу.	

Таблиця 5.20. Визначення основних меж встановлення цін

№ п/п	Основний рівень цін на товари-замінники	Основний рівень цін на товари-аналоги	Основний рівень доходів цільової групи користувачів	Верхня та нижня межі ціни
	Безкоштовні (нативні методи є безкоштовними та ідуть в комплектації з мовою програмування)	Їх не існує	Стабільно високий, щоб мати можливість використовувати сервіс підтримки та сервіс для обговорення нового функціоналу у повній його потужності	Продаж продукту – 1000$ за один екземпляр Підписка на оновлення – 100$ в місяць Додатковий пакет на support – 100$ в місяць Додатковий пакет на запит розробки додаткових функцій – 100$ в місяць

Таблиця 5.21. Формування основної системи збуту

№ п/п	Характеристика закупівельної поведінки користувачів	Функції збуту	Глибина каналу збуту	Система збуту
	Клієнти прагнуть купувати пакети на саппорт та на додаткові функції, оскільки вони зацікавлені у розвитку власного продукту	Billing, refund, підключення бібліотеки	1	Розповсюдження за сервісною моделлю з 1 альтернативнім каналом збуту

Таблиця 5.22. Концепція основних маркетингових комунікацій

№ п/п	Характеристика поведінки цільових клієнтів	Канали комунікацій для споживачів	Основні позиції для позиціонування	Завдання маркетингового повідомлення	Концепція маркетингового звернення
	Використання служби саппорту до повного інтегрування SDK	Telegram, slack, LinkedIn	Функціональність, інноваційність	Якомога найкраще донести до потенційних клієнтів, що вони потребують цей продукт	With our innovative processing methods you will never wait for your smartphone to finish its work

5.6. Створення програми розробки проекту

Фора: оптимізація часу обробки та навантаження на систему.

Рішення: розробити оптимізовані методі обробки великих об’ємів даних (XML-файлів) для різних ситуацій та представити їх у SDK. Таким чином інші

розробники, для яких придбають даний продукт, зможуть з легкістю маніпулювати вже створеними методами та швидко обробляти дані у своїх мобільних додатках.

Проблема: проблема полягає у тому, що на даний момент існують лише нативні методи та методи з деяких старих бібліотек, які дозволяють парсити XML-файли. Усі ці методи були призначені для файлів малих об'ємів і не використовують ніяких додаткових інструментів для оптимізації часу (наприклад, багатопоточність). Чим далі ми рухаємося у майбутнє, тим частіше ми зустрічаємося з великими обсягами даних. Таким чином, все більше Android розробників стикаються з проблемою парсингу таких даних. Стартап націлений на вирішення цієї проблеми.

Кроки:

- Розробка проекту – 01.03;
- Розробка методів та алгоритмів обробки – 01.04;
- Впровадження у готовий додаток – 01.06;
- Тестування ПЗ – 01.07;
- Просування продукту – 01.08;
- Остаточний реліз – 31.09.

Гіпотези: буде використовуватися для мобільних додатків для інтеграції с розумним будинком та оптимізує час обробки статистичних даних, які надсилаються власнику. Також має оптимізувати навантаження на систему.

Метрики:

- Час розробки: 6 місяців;
- Кількість перших продажів (платних підписок): 1000.

Потоки доходів:

- Інвестиції – 50 000$;
- Продаж продукту – 1000$ за один екземпляр;
- Підписка на оновлення – 100$ в місяць;

- Додатковий пакет на support – 100$ в місяць;
- Додатковий пакет на запит розробки додаткових функцій – 100$ в місяць.

Структура витрат:

- Оренда приміщення – 1000$ в місяць;
- З/П – 3000$ в місяць;
- Робочі станції – 6000$;
- Витрати на різні ресурси (GitHub, CI…) - 1000$ в місяць;
- Витрати на рекламу – 5000$.

Ключові партнери:

- GitHub – приватний репозиторій;
- Bitrise – CI для автоматизації процесів;
- Jira – для постановки задач.

Ключові види діяльності: розробка ПЗ (програмування).

Ключові ресурси: інтелектуальні (розробка алгоритмів та методів).

Ціннісні пропозиції: пропонується SDK з новітніми методами парсингу даних, які гарно оптимізовані за часом та не несуть великої навантаженості на систему.

Взаємини з клієнтами: зворотній зв’язок (оцінка та відклик клієнтів про продукт).

Канали збуту: компанії-розробники додатків для інтеграції з розумними будинками.

Споживчий сегмент: власники систем розумних будинків.

Таблиця 5.23. Календарний план-графік реалізації стартап-проекту

№ з/п	Етапи реалізації	1-й м.	2-й м.	3-й м.	4-й м.	5-й м.	6-й м.
1	Дослідження методів реалізації						
2	Видача технічного завдання						
3	Розподіл обов’язків						
4	Проектування алгоритмів обробки XML-файлів						
5	Розробка алгоритмів обробки XML-файлів						
6	Розробка SDK						
7	Інтеграція раніше розроблених алгоритмів у додаток						
8	Тестування ПЗ						
9	Пошук інвесторів						
10	Просування продукту						
11	Технічний супровід користувачів						
12	Реліз						

Таблиця 5.24. Планова потреба у виробничих площах

№ з/п	Тип приміщення (будівлі, ділянки, споруди)	Кількість одиниць	Площа, кв. м	Вимоги до приміщення (будівлі, ділянки, споруди)	Умови надання	Вартість, тис. грн.
1.	Опен-спейс на 3-х осіб	1	20	Великі вікна, гарне освітлення, наявність WI-FI та розеток у великій кількості	Оренда	25 тис. грн.в місяць
Разом		1	20	—	—	150 тис. грн.

Таблиця 5.25. Планова потреба у виробничому обладнанні та устаткуванні

№ з/п	Вид обладнання (устаткування, пристрою)	Тип (модель)	Виробник обладнання (устаткування, пристрою)	Терміни постачання	Вартість, тис. грн.
1.	Мобільна робоча станція	Macbook Pro 2019 15”	Apple	2 дні	60 тис. грн.за екземпляр
Разом:		—	—	—	180 тис. грн.

Таблиця 5.26. Планова вартість нематеріальних активів

№ з/п	Вид активів	Активи, що можуть бути віднесені до даного виду	Вартість, тис. грн.
1.	Права користування майном	- 3 стола, 3 стула, 3 розетки, 3 подовжувачі та шкаф для обягу	Відноситься до оренди
2.	Інші активи	Розміщення коду на GitHub	6
		Використання CI Bitrise	10
		Використання Maven репозиторіїв	8

Таблиця 5.27. Плановий обсяг виробництва продукції стартап-проекту

Вид продукції	Одиниця виміру	Обсяги виробництва за період		
		Перший рік	Другий рік	Третій рік
Нова версія	Реліз	4	6	6

Таблиця 5.28. Планова потреба у матеріальних ресурсах та комплектуючих

№ з/п	Вид ресурсу	Одиниця виміру	Витрати на одиницю продукції в натуральних показниках	Вартість на одиницю продукції, тис. грн.	Вартість за плановим обсягом виробництва за період, тис грн.		
					Перший рік	Другий рік	Третій рік
Всього матеріалів		—	—				
2.	Комплектуючі						
2.1.	Оперативна пам’ять для макбуків	Шт.	—	1.5	0	4.5	4.5
2.2	Клавіатура для макбуків	Шт.	—	5	0	15	15
Всього сировини				—	0	19.5	19.5
Разом:		—	—		0	19.5	19.5

Таблиця 5.29. Планова потреба та витрати на персонал

№ з/п	Категорія персоналу	Чисельність	Заробітна плата, тис грн. на місяць	Відрахування на соціальні заходи, тис грн. на місяць	Витрати на оплату праці за період, тис. грн.		
					Перший рік	Другий рік	Третій рік
1.	Адміністративний персонал (працівники апарату управління)						
1.1.	Проджект менеджер	1	25	12.5	450	517	595
1.2	Маркетолог	1	20	10	360	414	476

Закінчення таблиці 5.29

Всього витрати на оплату праці адміністративного персоналу			45	22.5	810	931	1071
2.	Промислово-виробничий персонал						
2.1.	Девелопер	1	30	15	540	621	714
Всього витрати на оплату праці промислово-виробни-чого персоналу			30	15	540	621	714
Разом:			75	37.5	1350	1552	1785

Таблиця 5.30**.** Загальні початкові витрати проекту

№ з/п	Стаття витрат	Обсяги витрат в 0-й рік, тис. грн.
1	Дослідження методів реалізації	40.5
2	Видача технічного завдання	40.5
3	Розподіл обов’язків	16.2
4	Проектування алгоритмів обробки XML-файлів	44.8
5	Розробка алгоритмів обробки XML-файлів	81
6	Розробка SDK	81
7	Інтеграція раніше розроблених алгоритмів у додаток	71.5
8	Тестування ПЗ	20.5
9	Пошук інвесторів	20.5
10	Просування продукту	20.5
11	Технічний супровід користувачів	6
12	Реліз	20
Разом		463

Таблиця 5.31. Планові загальногосподарські витрати

№ з/п	Стаття витрат	Витрати за період, тис. грн.		
		Перший рік	Другий рік	Третій рік
1.	Витрати на оренду земельних ділянок, будівель, приміщень, споруд	300	300	300
2.	Витрати на обладнання, устаткування та пристрої	180	19.5	19.5
3.	Витрати на придбання нематеріальних активів	288	288	288
4.	Витрати на персонал (на відрядження, соціальні заходи тощо)	1350	1552	1785
5.	Витрати на зв'язок	—	—	—
6.	Витрати на паливо та електроенергію	—	—	—
7.	Витрати на водопостачання	—	—	—
8.	Витрати на утримання обладнання та приміщень	—	—	—
9.	Витрати на збут	—	—	—
10.	Витрати на просування та рекламу	150	100	50
11.	Оплата юридичних послуг	—	—	—
12.	Податкові платежі (земельний, комунальний податки, інші)	—	—	—
Разом:		2268	2259.5	2442.5

Організаційну структура підприємства можна побачити на рис. 5.1.

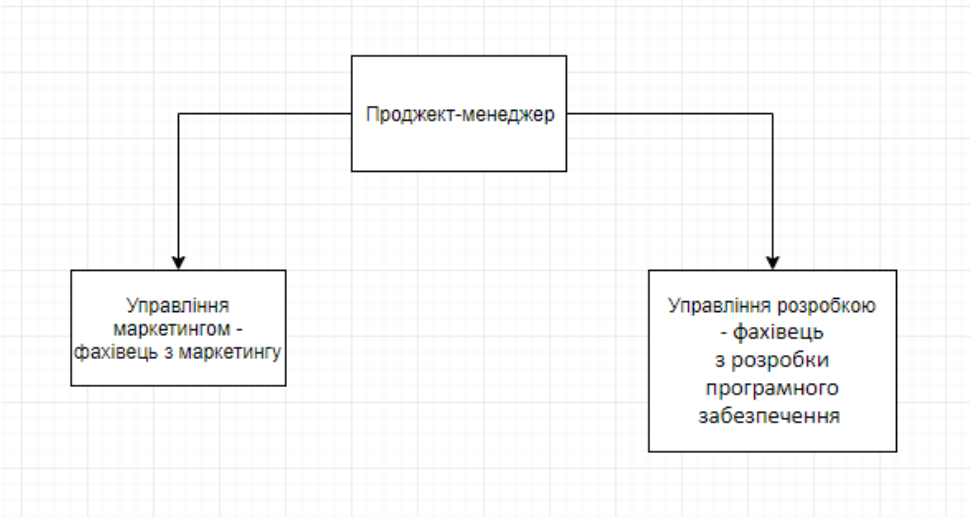


Рисунок 5.1 – Організаційна структура

Регламентуючі документи:

- Контракт прівника з підприємством (ЗП, вихідні, відпустки, кількість робочих годин на тиждень);
- Документи (звіти, довідки, списки) страхування працівників;
- Розрахунково-платіжні відомості (особові рахунки) працівників.

Стимул для працівників:

- Матерільний: підвищення ЗП щороку на 15%;
- Нематерільний: зазначений для кожного працівника у табличці нижче. До всього цього також можна віднести безкоштовні білети на ІТ-конференції та участь у відрядженнях.

Таблиця 5.32. Проджект менеджер

Критерій	Зміст
Основна освіта	Не має значення, яку саме освіту здобував проджект менеджер у минулому
Додаткова освіта, спеціалізація	Не має значення
Необхідний досвід роботи	3 роки у сфері ІТ (проджект менеджером)
Завдання	Створювати ТЗ, керувати командою, розподіляти завдання, брати участь у архітектурних та організаційних моментах
Знання	Функціонування мобільних додатків та розумних домів
Навички, вміння, ділові якості	Вміння спілкуватися з командою, доводити спрінт до кінця, стимулювати колег до роботи
Особистісні якості	Дружелюбний
Мотивація (що можемо запропонувати)	Лідерство у цікавому проекті

Таблиця 5.33. Маркетолог

Критерій	Зміст
Основна освіта	Закінчення маркетингового факультету
Додаткова освіта, спеціалізація	Не має значення
Необхідний досвід роботи	2 роки у сфері маркетингу та реклами
Завдання	Рекламувати розроблений продукт, знаходити відмінності від існуючих конкурентів там вміти приваблювати клієнтів
Знання	Маркетинг у IT-сфері
Навички, вміння, ділові якості	Вміння спілкуватися з командою, не боятися розпитувати колег про продукт, шоб краще його зрозуміти
Особистісні якості	Дружелюбний, вміння викладати суть у малі тексти
Мотивація (що можемо запропонувати)	Знайомство з можливими міжнародними партнерами

Таблиця 5.34. Розробник

Критерій	Зміст
Основна освіта	Технічна освіта
Додаткова освіта, спеціалізація	Проходження спеціальних курсів (не обов’язково)
Необхідний досвід роботи	4 роки в мобільній розробці (Android)
Завдання	Робота з багатопоточністю, вміння проектувати, робота с обрабкою великих масивів даних
Знання	Android development, Java, Kotlin, RxJava, Kotlin Coroutines, Threads
Навички, вміння, ділові якості	Вміння спілкуватися з командою, доведення діла до кінця
Особистісні якості	Дружелюбний, працьовитий, винахідливий
Мотивація (що можемо запропонувати)	Робота над цікавим та інноваційним рішенням

Висновок до розділу

У цьому розділі розроблено та описано маркетингові стратегії та підходи з розробки стратап-проекту. Також було визначено багато параметрів: попит, рентабільність і динаміка ринку, групи клієнтів, можливість та бар'єри входження, характеристику конкурентів та опонентів, конкурентоспроможність стартапу, альтернативні впровадження проекту.

ВИСНОВКИ

У цій магістерській дисертації розглядається, що таке МОС Андроїд, XML файли, для яких потреб вони використовуються, яка їх основна перспектива в майбутньому. Також описано, чому майбутнє IT стоїть прямо за додатками для мобільних операційних систем. Розглянуто приклади використання великих обсягів XML-документів та проблеми з їх парсингом на мобільних пристроях. Далі описується предмет дослідження, об'єкт дослідження, проблема і ставиться детальне завдання. Також розглядаються можливі аналоги семантичного аналізу файлів XML на платформі Андроїд, які не вирішують проблему.

Було вирішено використовувати концепцію багатопоточності, щоб знайти рішення проблеми. Було розглянуто, які типи багатопотокового програмування існують, які моделі та схеми використовуються для розробки додатків з цим типом програмування. Потім були обрані основні бібліотеки, які дозволяють нам використовувати багатопоточність для наших цілей. Серед них RxJava, різьблення і Kotlin Coroutines.

Розроблено SDK, у якому реалізовано 8 методів семантичного аналізу: .

- З implementation Threads в один потік;
- З implementation Threads в два потоки;
- З implementation Kotlin Coroutines в один потік;
- З implementation Kotlin Coroutines в два потоки;
- З implementation Kotlin Coroutines в чотири. потоки;
- З implementation RxJava в один потік;
- З implementation RxJava в два потоки; .
- З implementation RxJava і метода *parallelStream().*

Ця бібліотека інтегрована в додаток з зрозумілим та простим інтерфейсом. Після завершеної аналізу відображається повідомлення про втрачений час і кількість символів в документі (обраний параметр з простим списком всіх символів).

Щоб отримати XML файли з зовнішнього носія, запит на Андроїд системи використовується для отримання дозволу. Для читання вмісту файлів розроблений алгоритм, описаний в третьому розділі випускного проекту. З його допомогою ви можете легко ходити відповідно до даних в документі.

Після застосування програми, багато тестів проводяться для аналізу ефективності створених методів. Для такої оцінки використовуються два фактори: ресурсоємність методів і час, на який вони закінчують обробку. Певна залежність від використання n-ої кількості потоків для споживання системних ресурсів. Було також встановлено, що кожен метод може бути відповідним для різних обставин і потреб клієнтів програми. Також уважно слід розглядати метод обробки за допомогою RxJava і метод parallelStream(), так як вона безпосередньо залежить від багатоядерності гаджет, на якому виконується додаток.

У кінці було розроблено алгоритм (див. Додаток Є) для пошуку найоптимальнішого методу серед створених у SDK та все існуючих і розглянутих у цій дисертації. За допомогою алгоритму вдалося знайти місце для всіх методів за конкретних кейсів (завантаженість системи, кількість ресурсів гаджета, заряд батареї та деякі вхідні дані алгоритму). Серед головних умов було розглянуто такі:

- «Чи може алгоритм обробки навантажувати пристрій?»;
- «Чи наявний в девайсі багатоядерний процесор?»;
- «Чи цікавить нас найшвидший метод?»;
- «Чи великий за об’ємом файл?»;
- «Скільки пристрій може виділити пам’яті на один процес?»;
- «Кількість ядер»;
- «Чи навантажена система у даний момент?»;
- «Заряд батареї».

ПЕРЕЛІК ПОСИЛАНЬ

1. The Art of Multiprocessor Programming / Morgan Kaufmann, Maurice Herlihy, Nir Shavit 2012. – 536 с.
2. Java Thread 3rd edition / Scott Oaks, Henry Wong, 2004. – 362 с.
3. Java Concurrency in Practice / Joshua Bloch, Joseph Bowbeer, David Holmes, Doug Lea, 2006. – 432 с.
4. Concurrent Programming in Java: Design Principles and Patterns / Doug Lea, 1999. – 432 с.
5. Programming Concurrency on the JVM: Mastering Synchronization, STM, and Actors / Venkat Subramaniam, 2011. – 298 с.
6. Реактивное программирование с использованием RxJava / Нуркевич, Кристенсен, 2017. – 258 с.
7. Learning Concurrency in Kotlin / Miguel Angel Castiblanco Torres, 2018. – 512 с.
8. RxJava Essentials / Ivan Morgillo, 2015. – 132 с.
9. Android. Программирование для профессионалов / Билл Филлипс, Крис Стюарт, Кристин Марсикано, 2016. – 688 с.
10. SAX2 / Дэвид Браунелл, 2002. – 210 с.
11. Parsing XML data in Android Apps [Електронний ресурс] – 2018. – Режим доступу до ресурсу: https://medium.com/@ssaurel/parsing-xml-data-in-android-apps-71ef607fbb16.
12. Парсеры XML[Електронний ресурс] – 2013. – Режим доступу до ресурсу: http://developer.alexanderklimov.ru/android/theory/xmlparsers.php.
13. Многопоточность в Java [Електронний ресурс] – 2012 . – Режим доступу до ресурсу: https://habr.com/ru/post/164487/.
14. Параллельные потоки [Електронний ресурс] – 2018. – Режим доступу до ресурсу: https://metanit.com/java/tutorial/10.9.php.

15. Многопоточное программирование и его проблемы [Електронний ресурс] – 2017. – Режим доступу до ресурсу: https://geekbrains.ru/posts/multithreading_series.

ДОДАТОК А

Публікація «Огляд багатопоточності та бібліотек для роботи з нею на мобільній платформі Android»

Посилання на збірник:

https://novaosvita.com/wp-content/uploads/2020/12/IntWorldScProc-Kyiv-Nov2020.pdf

Сертифікат:

ДОДАТОК Б

Програмний код

```
class MainActivity : Activity() {

    val RC_ALL_PERMISIONS = 100

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        btnThreads.setOnClickListener {
            if (checkPermission()) {
                navigateTo(ThreadsActivity::class.java)
            }
        }
    }

    private fun navigateTo(activity: Class<ThreadsActivity>) =
startActivity(Intent(this, activity))

    private fun checkPermission(): Boolean {
        var result = false

        val sdCardWrite = Manifest.permission.WRITE_EXTERNAL_STORAGE
        val sdCardRead = Manifest.permission.READ_EXTERNAL_STORAGE
        val permissionSdCardWriteCheck =
ContextCompat.checkSelfPermission(this, sdCardWrite)
        val permissionSdCardReadCheck =
ContextCompat.checkSelfPermission(this, sdCardRead)
        if (permissionSdCardWriteCheck == PackageManager.PERMISSION_GRANTED
&& permissionSdCardReadCheck == PackageManager.PERMISSION_GRANTED) {
            result = true
        } else {
            requestPermissions(arrayOf(sdCardWrite, sdCardRead),
                    RC_ALL_PERMISIONS)
            result = false
        }

        return result
    }

    override fun onRequestPermissionsResult(requestCode: Int, permissions:
Array<out String>?, grantResults: IntArray?) {
        super.onRequestPermissionsResult(requestCode, permissions,
grantResults)
        if (requestCode == RC_ALL_PERMISIONS) {
            if (grantResults!![1] == PackageManager.PERMISSION_GRANTED) {
                navigateTo(ThreadsActivity::class.java)
            }
        }
    }
}


class ThreadsActivity : Activity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_threads)
        btnRun1.setOnClickListener {
```

```
            EventBus.getDefault().post(StartMessage())
            Handler().postDelayed({
                threadsSequentialWordCount()
            }, 500)
        }
        btnRun2.setOnClickListener {
            EventBus.getDefault().post(StartMessage())
            Handler().postDelayed({
                threadsTwoThreadsWordCount()
            }, 500)
        }
        btnRun3.setOnClickListener {
            EventBus.getDefault().post(StartMessage())
            rxJavaSequentialWordCount()
        }
        btnRun4.setOnClickListener {
            EventBus.getDefault().post(StartMessage())
            rxJavaTwoThreadsWordCount()
        }
        btnRun5.setOnClickListener {
            EventBus.getDefault().post(StartMessage())
            coroutinesSequentialWordCount()
        }
        btnRun6.setOnClickListener {
            EventBus.getDefault().post(StartMessage())
            coroutinesTwoThreadsWordCount()
        }
        btnRun7.setOnClickListener {
            EventBus.getDefault().post(StartMessage())
            coroutinesBetterWordCount()
        }

        btnRun8.setOnClickListener {
            EventBus.getDefault().post(StartMessage())
            rxJavaParallelStreamWordCount()
        }
    }

    override fun onStart() {
        super.onStart()
        EventBus.getDefault().register(this)
    }

    override fun onStop() {
        super.onStop()
        EventBus.getDefault().unregister(this)
    }

    @Subscribe(sticky = false, threadMode = ThreadMode.MAIN)
    fun onEvent(message: StartMessage) {
        LoadingViewMaterial.show(this)
    }

    @Subscribe(sticky = false, threadMode = ThreadMode.MAIN)
    fun onEvent(message: FinishMessage) {
        LoadingViewMaterial.hide()
    }

    @Subscribe(sticky = false, threadMode = ThreadMode.MAIN)
    fun onEvent(message: StaticticMessage) {
        Toast.makeText(this, "Words count: ${message.count} \nTime:
${message.time} millis", Toast.LENGTH_LONG).show()
    }
```

```
    //Threads and Locks Examples
    private fun threadsSequentialWordCount() {
        ThreadSequentialWordCount().run()
    }

    private fun threadsTwoThreadsWordCount() {
        ThreadTwoThreadsWordCount().run()
    }


    //RxJava 2 Examples
    private fun rxJavaSequentialWordCount() {
        RxSequentialWordCount().run()
    }

    private fun rxJavaTwoThreadsWordCount() {
        RxTwoThreadsWordCount().run()
    }

    private fun rxJavaParallelStreamWordCount() {
        RxParallelStreamWordCount().run()
    }


    //Kotlin Coroutines Examples
    private fun coroutinesSequentialWordCount() {
        CoroutinesSequentialWordCount().run()
    }

    private fun coroutinesTwoThreadsWordCount() {
        CoroutinesTwoThreadsWordCount().run()
    }

    private fun coroutinesBetterWordCount() {
        CoroutinesBetterWordCount(Source()).run()
    }
}


class ThreadSequentialWordCount {
    private val LOG_TAG = ThreadSequentialWordCount::class.java.canonicalName

    private val counts: HashMap<String, Int?> = HashMap()

    fun run() {

        val thread = Thread {
            val pagesOne = Pages(0, 700, Source().wikiPagesBatchOne())
            pagesOne.forEach { page -> Words(page.text).forEach {
countWord(it) } }

            val pagesTwo = Pages(0, 700, Source().wikiPagesBatchTwo())
            pagesTwo.forEach { page -> Words(page.text).forEach {
countWord(it) } }
        }

        val time = measureTimeMillis {
            thread.start()
            thread.join()
        }

        Log.d(LOG_TAG, "Number of elements: ${counts.size}")
        Log.d(LOG_TAG, "Execution Time: $time ms")
```

```
        EventBus.getDefault().post(FinishMessage())
        EventBus.getDefault().post(StaticticMessage(time, counts.size))
    }

    private fun countWord(word: String) {
        when (counts.containsKey(word)) {
            true -> counts[word] = counts[word]?.plus(1)
            false -> counts[word] = 1
        }
    }
}


class ThreadTwoThreadsWordCount {
    private val LOG_TAG = ThreadTwoThreadsWordCount::class.java.canonicalName

    private val counts: ConcurrentHashMap<String, Int?> = ConcurrentHashMap()

    fun run() {
        val threadOne = Thread {
            val pagesOne = Pages(0, 700, Source().wikiPagesBatchOne())
            pagesOne.forEach { page -> Words(page.text).forEach {
countWord(it) } }
        }

        val threadTwo = Thread {
            val pagesTwo = Pages(0, 700, Source().wikiPagesBatchTwo())
            pagesTwo.forEach { page -> Words(page.text).forEach {
countWord(it) } }
        }

        val time = measureTimeMillis {
            threadOne.start()
            threadTwo.start()
            threadOne.join()
            threadTwo.join()
        }

        Log.d(LOG_TAG, "Number of elements: ${counts.size}")
        Log.d(LOG_TAG, "Execution Time: $time ms")
        EventBus.getDefault().post(FinishMessage())
        EventBus.getDefault().post(StaticticMessage(time, counts.size))
    }

    private fun countWord(word: String) =
            counts.merge(word, 1) { oldValue, _ -> oldValue.plus(1) }
}


class RxSequentialWordCount {
    private val LOG_TAG = RxSequentialWordCount::class.java.canonicalName

    private val counts: HashMap<String, Int?> = HashMap()

    fun run() {
        val startTime = System.currentTimeMillis()

        val observable = Observable.fromCallable {
            var pagesOne = Pages.empty()
            val pagesOneCreationTime = measureTimeMillis {
                pagesOne = Pages(0, 700, Source().wikiPagesBatchOne())
            }
            pagesOne.forEach { page -> Words(page.text).forEach {
```

```
countWord(it) } }

            var pagesTwo = Pages.empty()
            val pagesTwoCreationTime = measureTimeMillis {
                pagesTwo = Pages(0, 700, Source().wikiPagesBatchTwo())
            }
            pagesTwo.forEach { page -> Words(page.text).forEach {
countWord(it) } }

            logPagesCreationTime(pagesOneCreationTime, pagesTwoCreationTime)
        }

        observable
                .doOnComplete { logExecutionData(System.currentTimeMillis() -
startTime) }
                .subscribeOn(Schedulers.single())
                .subscribe()
    }

    private fun countWord(word: String) {
        when (counts.containsKey(word)) {
            true -> counts[word] = counts[word]?.plus(1)
            false -> counts[word] = 1
        }
    }

    private fun logPagesCreationTime(timePagesOne: Long, timePagesTwo: Long)
{
        Log.d(LOG_TAG, "PageOne creation time: $timePagesOne ms")
        Log.d(LOG_TAG, "PageOne creation time: $timePagesTwo ms")
        Log.d(LOG_TAG, "Total Execution Pages Creation:
${timePagesOne.plus(timePagesTwo)} ms")
    }

    private fun logExecutionData(time: Long) {
        Log.d(LOG_TAG, "Number of elements: ${counts.size}")
        Log.d(LOG_TAG, "Execution Time: $time ms")
        EventBus.getDefault().post(FinishMessage())
        EventBus.getDefault().post(StaticticMessage(time, counts.size))
    }
}


class RxTwoThreadsWordCount {
    private val LOG_TAG = RxTwoThreadsWordCount::class.java.canonicalName

    private val counts: ConcurrentHashMap<String, Int?> = ConcurrentHashMap()

    fun run() {
        val startTime = System.currentTimeMillis()

        val observablePagesOne = Observable.fromCallable {
            val pagesOne = Pages(0, 700, Source().wikiPagesBatchOne())
            pagesOne.forEach { page -> Words(page.text).forEach {
countWord(it) } }
        }.subscribeOn(Schedulers.newThread())

        val observablePagesTwo = Observable.fromCallable {
            val pagesTwo = Pages(0, 700, Source().wikiPagesBatchTwo())
            pagesTwo.forEach { page -> Words(page.text).forEach {
countWord(it) } }
        }.subscribeOn(Schedulers.newThread())
```

```
        observablePagesOne
                .mergeWith(observablePagesTwo)
                .doOnComplete { logData(System.currentTimeMillis() -
startTime) }
                .subscribe()
    }

    private fun countWord(word: String) =
            counts.merge(word, 1) { oldValue, _ -> oldValue.plus(1) }

    private fun logData(time: Long) {
        Log.d(LOG_TAG, "Number of elements: ${counts.size}")
        Log.d(LOG_TAG, "Execution Time: $time ms")
        EventBus.getDefault().post(FinishMessage())
        EventBus.getDefault().post(StaticticMessage(time, counts.size))
    }
}


class RxParallelStreamWordCount {
    private val LOG_TAG = RxParallelStreamWordCount::class.java.canonicalName

    fun run() {
        val startTime = System.currentTimeMillis()

        val singleLettersCount = Single.fromCallable {
            Pages(0, 700, Source().wikiPagesBatchOne())
                    .plus(Pages(0, 700, Source().wikiPagesBatchTwo()))
                    .parallelStream()
                    .flatMap {
StreamSupport.stream(Words(it.text).spliterator(), false) }
                    .collect(Collectors.groupingBy({ it.toString() },
Collectors.counting()))
        }

        singleLettersCount
                .subscribeOn(Schedulers.io())
                .subscribe { data ->
                    logData(System.currentTimeMillis() - startTime,
data.size)
                }
    }

    private fun logData(time: Long, count: Int) {
        Log.d(LOG_TAG, "Number of elements: $count")
        Log.d(LOG_TAG, "Execution Time: $time ms")
        EventBus.getDefault().post(FinishMessage())
        EventBus.getDefault().post(StaticticMessage(time, count))
    }
}


class CoroutinesSequentialWordCount {
    private val counts: HashMap<String, Int?> = HashMap()

    private val LOG_TAG =
CoroutinesSequentialWordCount::class.java.canonicalName

    fun run() {
        launch(newSingleThreadContext("myThread")) {
            val startTime = System.currentTimeMillis()
            counter()
            logData(System.currentTimeMillis() - startTime)
```

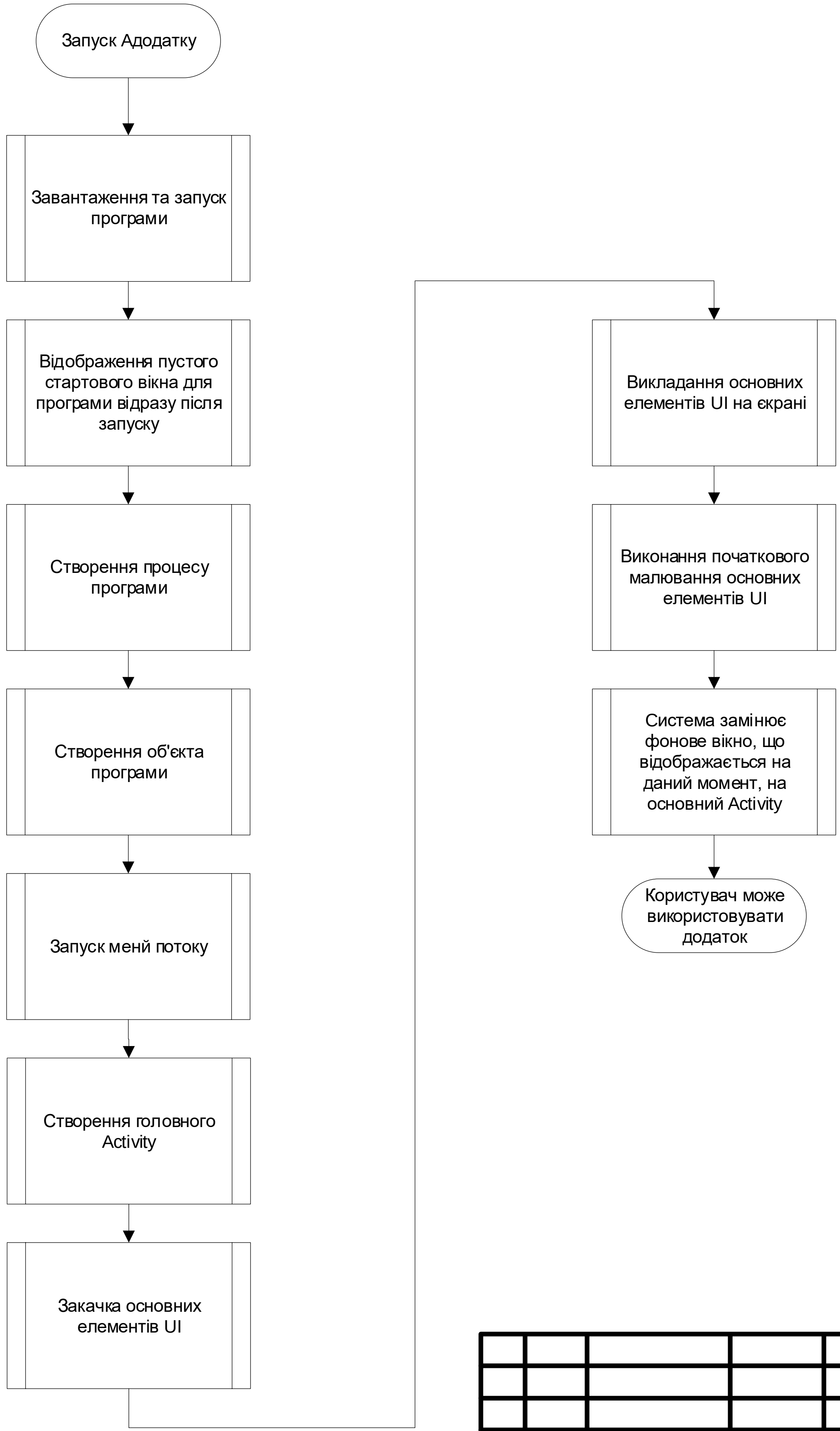
```
        }
    }

    private suspend fun counter() {
        val pagesOne = Pages(0, 700, Source().wikiPagesBatchOne())
        pagesOne.forEach { page -> Words(page.text).forEach { countWord(it) }
}

        val pagesTwo = Pages(0, 700, Source().wikiPagesBatchTwo())
        pagesTwo.forEach { page -> Words(page.text).forEach { countWord(it) }
}
    }

    private fun countWord(word: String) {
        when (counts.containsKey(word)) {
            true -> counts[word] = counts[word]?.plus(1)
            false -> counts[word] = 1
        }
    }

    private suspend fun logData(time: Long) {
        Log.d(LOG_TAG, "Number of elements: ${counts.size}")
        Log.d(LOG_TAG, "Execution Time: $time ms")
        EventBus.getDefault().post(FinishMessage())
        EventBus.getDefault().post(StaticticMessage(time, counts.size))
    }
}


class CoroutinesTwoThreadsWordCount {
    private val counts: ConcurrentHashMap<String, Int?> = ConcurrentHashMap()

    private val LOG_TAG =
CoroutinesTwoThreadsWordCount::class.java.canonicalName

    fun run() {
        val poolContext = newFixedThreadPoolContext(2, "ThreadPool")
        launch(poolContext) {
            val time = measureTimeMillis {
                val one = async(poolContext) { counterPages1() }
                val two = async(poolContext) { counterPages2() }
                one.await()
                two.await()
            }
            logData(time)
        }
    }

    private fun counterPages1() {
        val pagesOne = Pages(0, 700, Source().wikiPagesBatchOne())
        pagesOne.forEach { page -> Words(page.text).forEach { countWord(it) }
}
    }

    private fun counterPages2() {
        val pagesTwo = Pages(0, 700, Source().wikiPagesBatchTwo())
        pagesTwo.forEach { page -> Words(page.text).forEach { countWord(it) }
}
    }

    private fun countWord(word: String) =
            counts.merge(word, 1) { oldValue, _ -> oldValue.plus(1) }
```

```
    private fun logData(time: Long) {
        Log.d(LOG_TAG, "Number of elements: ${counts.size}")
        Log.d(LOG_TAG, "Execution Time: $time ms")
        EventBus.getDefault().post(FinishMessage())
        EventBus.getDefault().post(StaticticMessage(time, counts.size))
    }
}


class CoroutinesBetterWordCount(source: Source) {
    private val LOG_TAG = CoroutinesBetterWordCount::class.java.canonicalName

    private val filePagesOne = source.wikiPagesBatchOne()
    private val filePagesTwo = source.wikiPagesBatchTwo()

    fun run() {
        launch(CommonPool) {
            val time = measureTimeMillis {
                val one = async(CommonPool) { counter(0.rangeTo(2000),
filePagesOne) }
                val two = async(CommonPool) { counter(2001.rangeTo(3000),
filePagesOne) }
                val three = async(CommonPool) { counter(0.rangeTo(2000),
filePagesTwo) }
                val four = async(CommonPool) { counter(2001.rangeTo(3000),
filePagesTwo) }
                mergeResults(one.await(), two.await(), three.await(),
four.await())
            }
            logData(time)
        }
    }

    private fun counter(range: IntRange, file: File): HashMap<String, Int?> {
        val counts: HashMap<String, Int?> = HashMap()
        val pages = Pages(range.start, range.endInclusive, file)
        pages.forEach { page -> Words(page.text).forEach { countWord(counts,
it) } }
        return counts
    }

    private fun countWord(counts: HashMap<String, Int?>, word: String) =
            counts.merge(word, 1) { oldValue, _ -> oldValue.plus(1) }

    private fun logData(time: Long) {
        Log.d(LOG_TAG, "Execution Time: $time ms")
        EventBus.getDefault().post(FinishMessage())
        EventBus.getDefault().post(StaticticMessage(time, 196681))
    }

    @TargetApi(VERSION_CODES.N)
    fun <K, V> Map<K, V>.mergeReduce(other: Map<K, V>, reduce: (V, V) -> V =
{ _, b -> b }): Map<K, V> =
            this.toMutableMap().apply { other.forEach { merge(it.key,
it.value, reduce) } }
}
```

ДОДАТОК В

Процес холодного старту Android додатку

ІК-91.28 3132.004 ВЗ



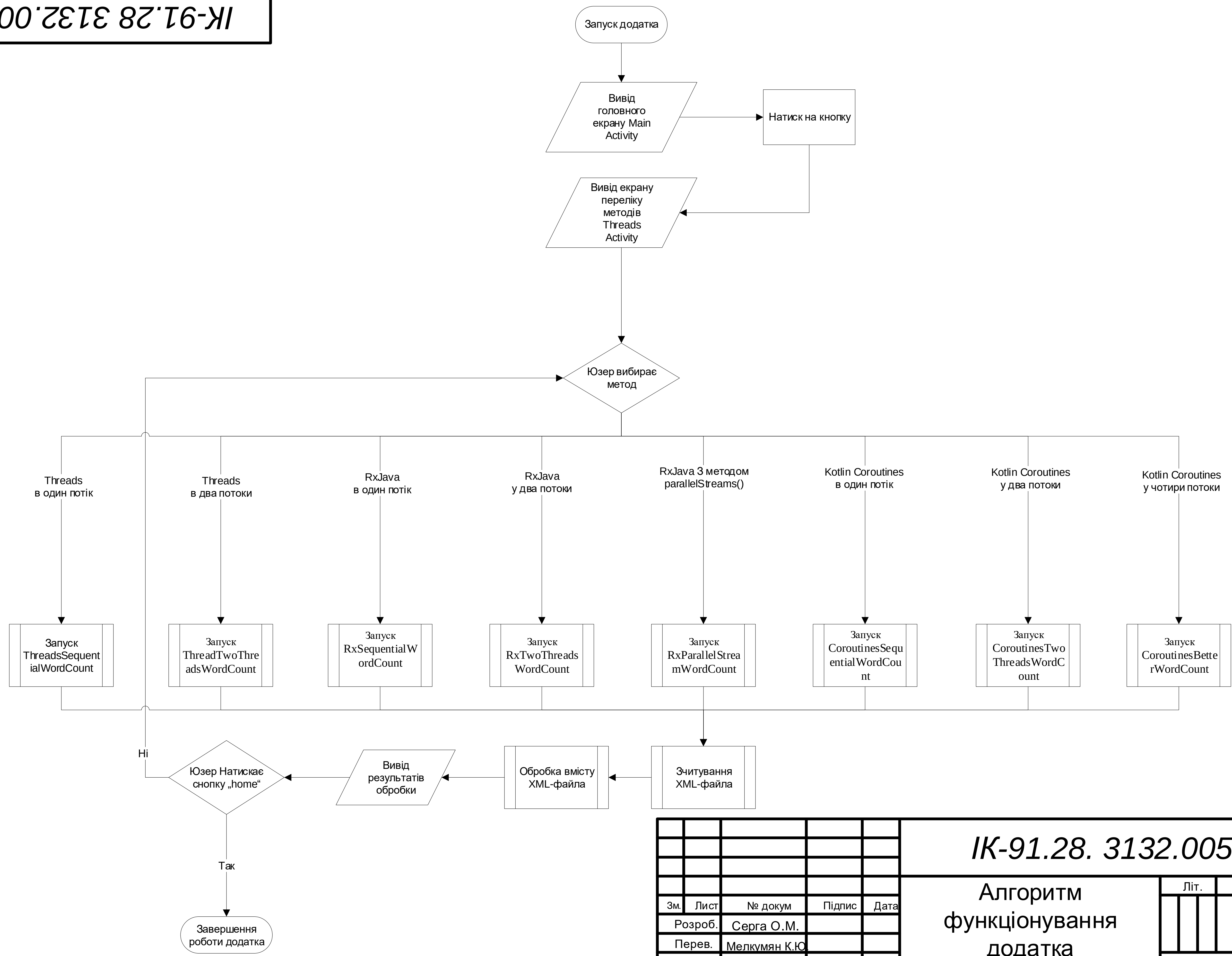
Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № оріг.

Зм.	Лист	№ докум	Підпис	Дата		
Розроб.		Серга О.М.			*ІК-91.28. 3132.004 ВЗ*	
Перев.		Мелкумян К.Ю			Процес холодного старту Android додатку	Літ. / Маса / Мірило
						Лист 1 / Листів 1
Н.контр		Пасько В.П			Кафедра Технічної кібернетики	Група ІК-91мп
Затв.		Пархомей І.Р.				

ДОДАТОК Г

Алгоритм функціонування додатка

ІК-91.28 3132.005 ВЗ

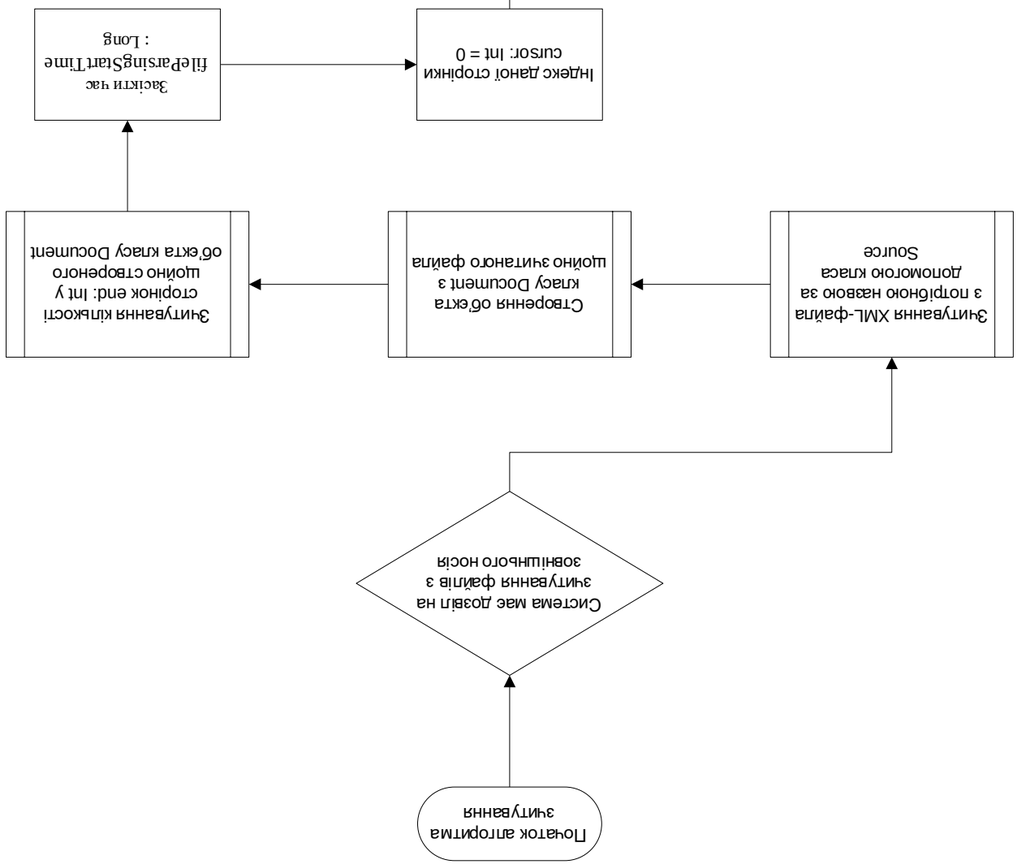


					ІК-91.28. 3132.005 ВЗ			
Зм.	Лист	№ докум	Підпис	Дата	Алгоритм функціонування додатка	Літ.	Маса	Мірило
Розроб.		Серга О.М.						
Перев.		Мелкумян К.Ю.						
						Лист 1	Листів 1	
Н.контр		Пасько В.П			Кафедра Технічної кібернетики	Група ІК-91мп		
Затв.		Пархомей І.Р.						

Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.

ДОДАТОК Ґ

Алгоритм зчитування XML-файла додатком



Зм.	Лист	№ докум	Підпис	Дата	
Розроб.		Сєра О.М.			
Перев.		Мелкумян К.Ю			
Н.контр		Пасько В.П			
Затв.		Пархомей І.Р.			

ІК-91.28. 3132.006 ВЗ

Алгоритм зчитування XML-файла додатком

Літ.	Маса	Масштаб

Лист 1 Листів 1

Кафедра Технічної кібернетики

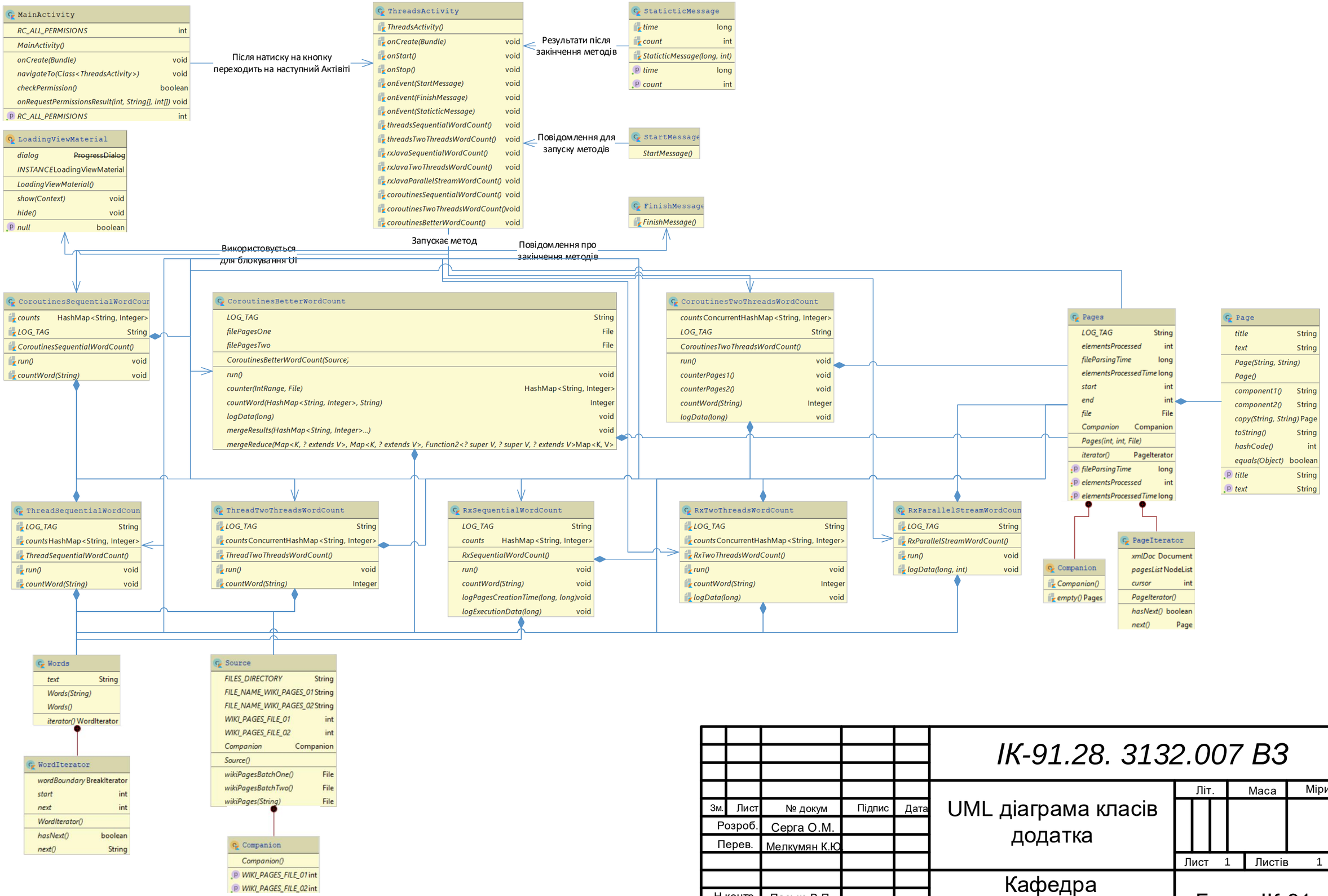
Група ІК-91мп

Інв. № орит.	Підпис і дата	Взам. інв. №	Інв. № дубл.	Підпис і дата

ІК-91.28 3132.006 ВЗ

ДОДАТОК Д

UML діаграма класів додатка



Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.

Зм.	Лист	№ докум	Підпис	Дата
Розроб.		Серга О.М.		
Перев.		Мелкумян К.Ю.		
Н.контр		Пасько В.П		
Затв.		Пархомей І.Р.		

ІК-91.28. 3132.007 ВЗ

UML діаграма класів додатка

Літ.	Маса	Мірило

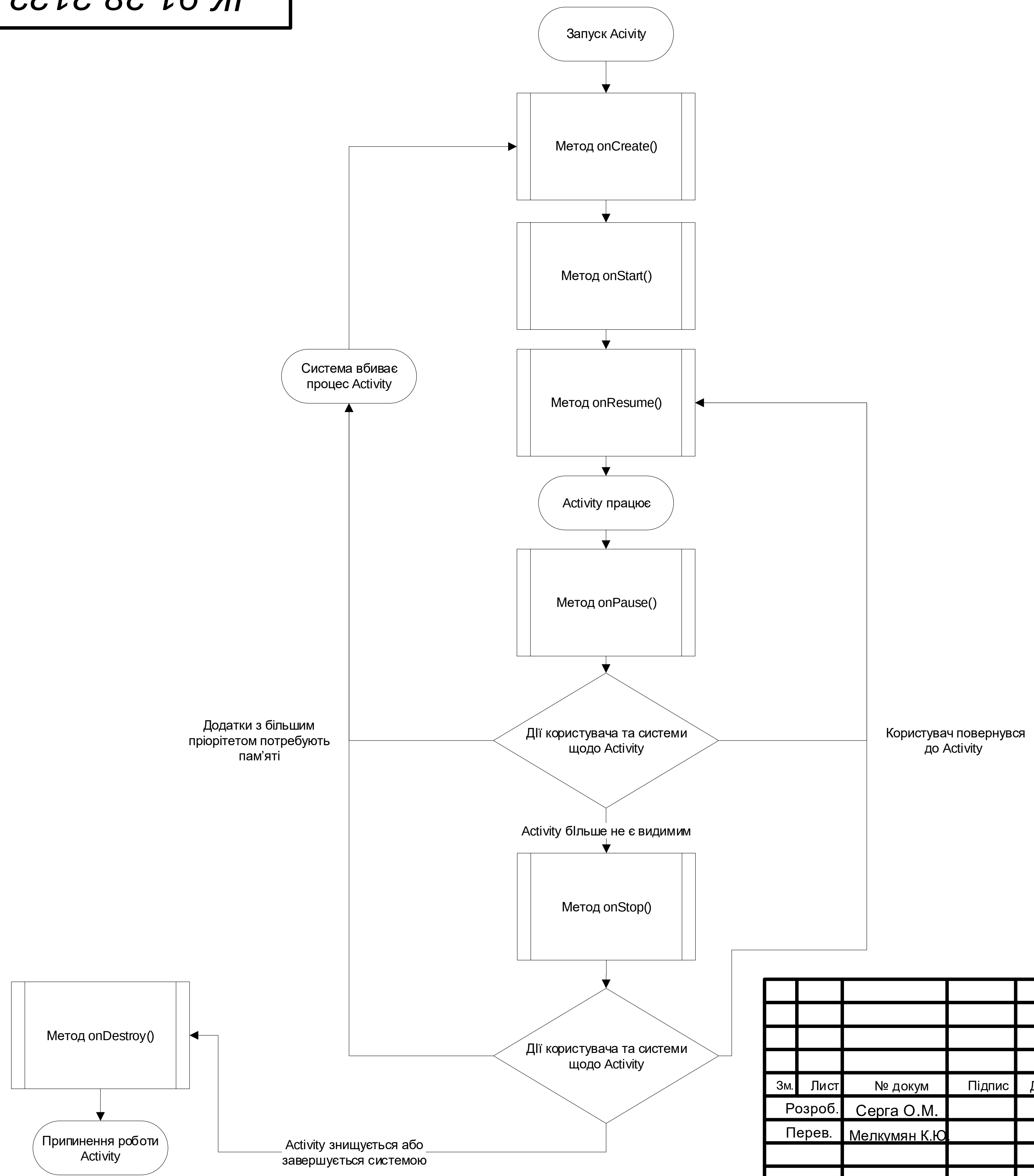
Лист 1 Листів 1

Кафедра Технічної кібернетики

Група ІК-91мп

ДОДАТОК Е

Життєвий цикл Activity



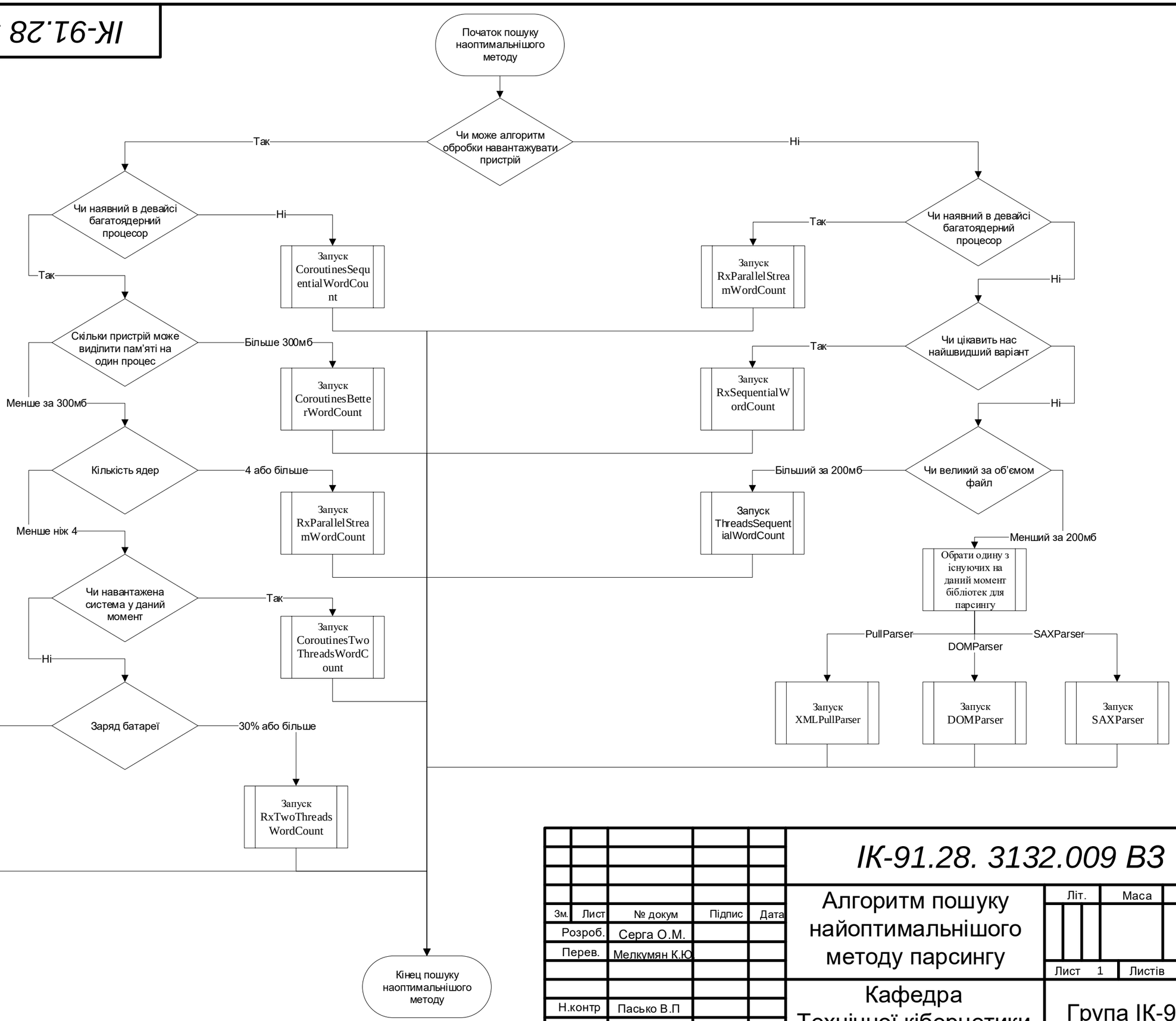
Зм.	Лист	№ докум	Підпис	Дата	ІК-91.28. 3132.008 ВЗ		
Розроб.		Серга О.М.			Життєвий цикл Activity	Літ. / Маса / Мірило	
Перев.		Мелкумян К.Ю				Лист 1	Листів 1
Н.контр		Пасько В.П			Кафедра Технічної кібернетики	Група ІК-91мп	
Затв.		Пархомей І.Р.					

Інв. № ориг. | Підпис і дата | Взам. інв. № | Інв. № дубл. | Підпис і дата

ІК-91.28 3132.008 ВЗ

ДОДАТОК Є

Алгоритм пошуку найоптимальнішого методу парсингу



Зм.	Лист	№ докум	Підпис	Дата	ІК-91.28. 3132.009 ВЗ		
Розроб.		Серга О.М.			Алгоритм пошуку найоптимальнішого методу парсингу	Літ.	Маса / Мірило
Перев.		Мелкумян К.Ю.				Лист 1	Листів 1
Н.контр		Пасько В.П			Кафедра Технічної кібернетики	Група ІК-91мп	
Затв.		Пархомей І.Р.					

Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.

ДОДАТОК Ж

Звіт результатів перевірки на унікальність

UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

Дата перевірки:
08.12.2020 08:10:34 EET

Дата звіту:
08.12.2020 08:13:44 EET

ID перевірки:
1005395885

Тип перевірки:
Doc vs Internet + Library

ID користувача:
76913

Назва документа: **Серга_ІК-91мп**

Кількість сторінок: **66** Кількість слів: **11625** Кількість символів: **83638** Розмір файлу: **119.98 KB** ID файлу: **1005687910**

3.5%
Схожість

Найбільша схожість: **1.75%** з джерелом з Бібліотеки (ID файлу: **1000776461**)

0.46% Джерела з Інтернету	48	Сторінка 68
3.5% Джерела з Бібліотеки	147	Сторінка 68

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел